STUDIES

ON

COMBINATORIAL OPTIMIZATION ALGORITHMS

NAOKI KATOH

STUDIES

ON

COMBINATORIAL OPTIMIZATION ALGORITHMS

NAOKI KATOH

JULY 1980

STUDIES

ON

COMBINATORIAL OPTIMIZATION ALGORITHMS

BY

NAOKI KATOH

Submitted in partial fulfillment of the

requirement for the degree of

DOCTOR OF ENGINEERING

at

Kyoto University

KYOTO, JAPAN

July 1980

PREFACE

Many mathematical programming problems found in various engineering fields contain a set of combinatorial constraints inherent to the problem structure. These problems can usually be formulated as combinatorial optimization problems. Solution techniques to solve combinatorial optimization problems are generally called combinatorial optimization algorithms. Research on combinatorial optimization algorithms has made a great progress in recent years in both theory and practice. As a result, efficient algorithms are now available for certain classes of combinatorial optimization problems. However, many important combinatorial optimization problems still do not have good algorithms. Moreover, even for the problems already having good algorithms, there still remains a large possibility that the existing good algorithms can be further improved.

In view of these situations, this thesis is devoted to propose efficient algorithms for solving several types of combinatorial optimization problems. The problems to be treated in this thesis are divided into two important categories. One is the graph optimization problem and the other is the resource allocation problem. The graph optimization problem is to find an optimal subgraph in a given graph (or to design an optimal graph) among those satisfying certain graphical constraints. The resource

allocation problem is to allocate a fixed amount of discrete resources to a fixed number of activities in an optimal way. Both of these problems are important members of combinatorial optimization problems.

To begin with, this thesis deals with K shortest simple path problem and K minimum spanning tree problem. The shortest path problem and the minimum spanning tree problem have long been investigated by many authors. Efficient algorithms have been also developed for the problems. However, in many practical situations, obtaining only one optimal solution is not sufficient but rather K best solutions are often needed. Thus both K shortest simple path problem and K minimum spanning tree problem are equally important. This thesis proposes efficient algorithms for these two problems, each of which improves the running time of the best known algorithms.

Secondly, this thesis deals with the well-known simple resource allocation problem. This problem has a rather long history, and some efficient algorithms have also been developed. A new algorithm, which is simple and efficient, is proposed in this thesis, based on the Lagrange multiplier method.

It should be noted, however, that the resource allocation problems arising in real world often have rather complicated constraints or require some other criteria. This necessitates further research on other types of the resource allocation

problem. In this respect, this thesis then deals with four types of the resource allocation problem, which can be formulated as generalizations or variants of the simple resource allocation problem. Some important properties of the problems are developed and they are utilized to yield efficient algorithms.

The research on efficient combinatorial optimization algorithms is a young field. Thus further investigation should be necessary. The author hopes that the work in the thesis will be helpful for further research in this growing field.

ACKNOWLEDGEMENTS

The author wishes to express his sincere appreciation to Professor H. Mine of Kyoto University for supervising this thesis. His constant encouragement and invaluable comments on the thesis have greatly helped to accomplish the work.

The author also wishes to thank Associate Professor T. Ibaraki of Kyoto University, who has always given eager suggestions and persistent discussions concerning the whole contents in the thesis. He also read the manuscript of the thesis and provided accurate comments, which are sincerely appreciated.

Furthermore, the author would like to express his gratitude to Professor T. Hasegawa of Kyoto University for his hearty encouragement. He also has provided helpful comments concerning the work in Chapters 2 and 8.

The author is also indebted to Dr. Y. Nomura and Dr. T. Kawamura and all his colleagues of the Center for Adult Diseases, Osaka for their earnest encouragements and generous support in the completion process of the thesis.

The author has to mention and gratefully acknowledge the comments, suggestions and cooperation offered by Associate Professor K. Ohno, Dr. H. Ishii, Dr. M. Fukushima, Dr. Y. Nakamori and Mr. K. Teranaka, while he was in Department of Applied Mathematics and Physics, Kyoto University. Moreover, the author is

CHAPTER 1

INTRODUCTION

1.1 Combinatorial Optimization Algorithms

Systems which appear in various engineering fields, especially information processing and operations research, often have a set of combinatorial constraints inherent to their structures and functions. Thus we are faced with combinatorial optimization problems which ask to obtain an optimal solution under some combinatorial constraints. It should be noted that combinatorial optimization problems usually have finitely many feasible solutions. Hence, enumeration can be in principle applied to find an optimal solution among all feasible solutions. However, the enumeration of all feasible solutions requires a prohibitively large amount of computation time, if done by the straightforward exhaustive method, and hence even moderate size problems often become computationally intractable. This necessitates the development of efficient combinatorial optimization algorithms.

Though combinatorial optimization problems can be, in general, formulated by using the concept of combinatorics, classical combinatorics mainly concerns with investigation of existence or enumeration of solutions satisfying certain combinatorial constraints, and hence it has not paid much attention to optimization problems. In addition, neither linear programming or nonlinear

programming, which forms a central portion of mathematical programming, deals with problems having combinatorial constraints.

Nevertheless, as the importance of combinatorial optimization problems increases, the research in this area has made a great progress. At present, a considerable portion of research efforts in mathematical programming are being devoted to combinatorial optimization problems.

It should be noted, however, that combinatorial optimization problems have characteristics quite different from the other mathematical programming problems. For example, in linear programming problems the simplex method has been proved to be efficient and useful. As to nonlinear programming problems, much research effort has been devoted to general solution techniques.

On the other hand, it is difficult to develop general solution techniques which work efficiently for all types of combinatorial optimization problems. Nevertheless some general solution techniques have been developed, i.e., integer programming [N1], sequential decision process [K2, I1, I2], dynamic programming [B5, D3] and branch-and-bound method [L2, I5, I6, I7, I8]. (Ibaraki [I9], for instance, covers a wide variety of recent results on sequential decision process, dynamic programming and branch-and-bound method.) However, it has been known for certain classes of combinatorial optimization problems that such techniques become computationally intractable as the problem size increases.

Moreover, the recent results on the theory of computational complexity [C4, G4, K3] provide us with some classes of combinatorial problems which do not probably have efficient algorithms. Even for tractable problems, such techniques are often less efficient than algorithms specially devised for solving only one class of combinatorial optimization problems.

In these respects, if a class of combinatorial optimization problems is to be solved, it is advisable to develop an efficient algorithm suited for the problem by taking advantage of the specific problem structure. Much effort to develop such algorithms has been recently made, especially for these ten years. As a result, many efficient algorithms are available for some classes of combinatorial optimization problems [A1, L4, T4]. At present, development of efficient combinatorial optimization algorithms is an important field in the research of combinatorial optimization problems, from both theoretical and practical viewpoints. It should be noted, however, that little has been known about whether the existing good algorithms can be further improved [A1, L4, T4]. Thus further research would be necessary in this growing field.

1.2 Efficiency Measures of Algorithms

This section discusses efficiency measures which are applied in analyzing combinatorial optimization algorithms to be proposed in this thesis. Running time and storage space have been considered in the literature as the most fundamental factors which specify the efficiency of algorithms. Since computers with large main memory now become available, running time is more important than storage space. The running time limits the size of the problems to be handled by the computers [A1, T4]. Hence, throughout this thesis the efficiency of algorithms is mainly measured by their running time.

If the running time required for an algorithm is expressed as a function T(n) of the problem size n, then T(n) is called the time complexity of the algorithm. It is noted that the asymptotic property of T(n) sometimes plays a more important role than the exact value of T(n) itself. Usually, the function form of T(n) contains some constant factors and coefficients. For example, let $T(n) = c_0 + c_1 n + c_2 n^2$, where c_0, c_1 and c_2 are constants. In this case, the behavior of T(n) is essentially determined by the main term $c_2 n^2$, as the problem size n asymptotically increases. This implies that the algorithm is said to have $O(n^2)$ running time.

It is noted, however, that, even if the problem size n is specified for a given problem, the function form of T(n) cannot

be uniquely determined, because there exist innumerable problem instances with size n. In order to determine the function form of T(n), it is necessary to introduce a certain measure for the evaluation of the algorithm. The following two measures have been proposed: average running time and worst-case running time. A worst-case running time guarantees the performance of an algorithm in the sense that the program requires no more time than the specified time bound for any problem. A worst-case running time sometimes provides time bounds which are too large in the practical sense. For example, the simplex method of linear programming requires an exponential worst-case running time [K15]. However, it has been experimentally known that it runs much faster than exponential. This means that an average running time might be more effective and realistic than the worst-case running time. In order to analyze the average running time, it is necessary to assume an appropriate probability measure on input data which specify the problem instances to be solved. However, for a realistic probability measure, it usually becomes very difficult to analyze the running time. Hence, this thesis concentrates on the worst-case running time.

The running time required for an algorithm is usually measured by CPU time. It depends on the machine and programming language by which the algorithm is implemented. In order to make the running time measure independent of the machine and

programming language actually used, some conceptual computation models have been proposed. Among them are Turing machine, RAM (random access machine), and RASP (random access stored program machine). The formal descriptions of these models are not given here (see [A1] for the details). This thesis analyzes running time of the algorithms based on RAM or RASP model, where one unit time is required for all fundamental operations such as addition, subtraction, multiplication, division, comparison and substitution. However, if algorithms are described by machine languages used in these models, it requires tremendous tedious tasks and loses the essence of the algorithms. Hence, this thesis uses the higher-level languages, such as ALGOL-like languages (see [A1]) or some other languages, which are easy to describe algorithms.

1.3 Purpose of the Thesis and Historical Background

The main purpose of this thesis is to propose efficient algorithms for certain types of combinatorial optimization problems, which are divided into two important categories. One is the graph optimization problem and the other is the resource allocation problem.

Many combinatorial problems are formulated in terms of graph, which is defined by a vertex set and an edge set. For example, scheduling problems in operations research, network analysis in electrical engineering, program segmentation in computer sciences, and analysis of Markov chains in probability theory - all of these can be formulated as graph theoretical problems. Thus it is worthwhile to investigate such problems formulated in terms of graph from the algorithmic point of view.

The resource allocation problem is to allocate a fixed amount of discrete resources to a fixed number of activities in an optimal way. This type of problems arise in many application areas such as budgeting problems, equipment investment problems, production planning problems, portfolio-selection problems, marketing problems (e.g., allocating an advertizing budget to a number of territories), load distribution among power generators and so forth (see for example [K4, L6, V1, V2]). Thus the resource allocation problem is also an important member of combinatorial

optimization problems.

The first part of this thesis proposes efficient algorithms for K best solutions in two types of famous graph optimization problems, i.e., the shortest path problem and the minimum spanning tree problem.

K best solutions of a combinatorial optimization problem have been required in many actual cases. For instance, combinatorial optimization problems which occur in real world do not usually have a simple structure, but have some complicated side constraints. In general, it is difficult to solve such actual problems, but easy to solve the simplified problem obtained by neglecting some of the complicated side constraints. In such cases, a best solution to the original problem should be found among the K best solutions found for the simplified problem.

In addition to this, it can often happen that a problem has more than one objective function. In such a multiobjective environment, admissible solutions may be found among the K best solutions to the single objective problem obtained by neglecting the additional objective functions.

In these respects, it is important to develop efficient algorithms for obtaining K best solutions in the standard combinatorial optimization problems. Actually, many authors have developed efficient algorithms for obtaining K best solutions in several types of combinatorial optimization problems, e.g.,

the shortest path problem, the minimum spanning tree problem, the optimum branching problem and the minimum assignment problem. In particular, the K shortest path problem has a long history. To the author's knowledge, this problem was first studied by Bellman and Kalaba[B4] in 1960. Since then, numerous papers have been published [C4, F2, H5, L1, M2, M3, P1-P3, S2-S4, W2, W4, W5, Y2, Y3]. In addition, the K minimum spanning tree problem has been studied by Burns and Half [B8], Camerini et al. [C1] and Gabow [G1], the K optimum branching problem by Camerini et al. [C2], and the K minimum assignment problem by Murty [M6].

The proposed efficient algorithms in the thesis for the K shortest path problem and the K minimum spanning tree problem improve the efficiency of the best known algorithms.

The second part of the thesis deals with several types of the resource allocation problem. The resource allocation problem was first studied by Koopman [K17] in 1952. Since then, numerous papers have been published (Ibaraki[I4] includes a lot of relevant references). Dynamic programming (DP) is one of the methods for solving the resource allocation problems (see textbooks Dreyfus and Law[D3] and Wagner[W1]). In applying DP, its multi-stage decision process can be represented by a directed graph. Thus the resource allocation problem can be regarded as the shortest path problem, since an optimal solution is obtained by finding the shortest path in the graph. Besides this approach, however,

more efficient algorithms have been known for certain types of the resource allocation problem (see [I4]).

In the second part of the thesis, the well-known simple resource allocation problem is first treated. This problem is to minimize the sum of separable convex functions under the constraint that the sum of integer variables is equal to a given integer, and it has been extensively studied by many authors (see [K17, G5, D4, E1, F1, F4, G2, H2, K1, M1, M4, P6, S5]). Some efficient algorithms have been also developed. This thesis proposes a new efficient algorithm for this problem.

It should be noted that the resource allocation problems which appear in real world do not usually have such a simple objective function (or constraint) as the above simple resource allocation problem. In this respect, this thesis then investigates new types of the resource allocation problem which frequently arise in many application areas such as the optimal sample size problem to estimate the urban air pollution [I10] and the apportionment problem [B1, B2, B3]. These problems are mathematically formulated as variants or generalizations of the simple resource allocation problem. They are described as follows.

(i) A generalized problem obtained from the simple resource allocation problem by allowing more than one constraint.

(ii) A variant problem obtained by exchanging the role of objective function and constraint.

(iii) An equipollent resource allocation problem which is

to minimize the maximum difference of the resulting profits between activities under the same constraint as the simple resource allocation problem.

(iv) A problem of obtaining K best solutions in the simple resource allocation problem.

This thesis proposes an efficient algorithm for each of the above problems except the problem (i). In particular, the problem (i) is proved to be inherently difficult.

1.4 Outline of the Thesis

This thesis consists of ten chapters. Chapters 2 through 9 propose efficient algorithms for several types of combinatorial optimization problems.

Chapter 2 deals with K shortest simple path problem. This problem has been extensively studied by many authors, i.e., [C4, L1, P1, P2, P3, W2, Y2, Y3]. Especially, Yen's algorithm [Y2, Y3] is the fastest one with $O(Kn^3)$ running time, where n is the number of vertices in a graph. This chapter presents a more efficient algorithm with $O(Kn^2)$ running time.

Chapter 3 deals with K minimum spanning tree problem, which has been studied by [B8, C1, G1]. The best known algorithm [G1] requires $O(Km\,\alpha(m,n) + m \log n)$ running time and $O(K+m)$ storage space, where n and m are the numbers of vertices and edges in a graph respectively, and α is Tarjan's inverse of Ackermann function [T2]. This chapter proposes a more efficient algorithm with $O(Km + \min(m \log\log n, n^2))$ time and $O(K+m)$ space.

Chapters 4-9 deal with several types of the resource allocation problems. First, Chapter 4 considers the simple resource allocation problem. In addition to DP approach, some algorithms have been known for this problem [F1, G5, S5]. This chapter presents a simple and efficient algorithm based on the Lagrange multiplier method, requiring $O(n^2 \log^2 N)$ running time, where n is the number of activities and N is the amount of resources.

Chapter 5 generalizes the above simple resource allocation problem by introducing more than one resource constraint. This generalization has many applications (i.e., budgeting problems and investment problems of weapon systems). This chapter proves that one type of the generalized problem can be reduced to the simple resource allocation problem discussed in Chapter 4 by applying a certain transformation ; the resulting problem can be easily solved. However, most of the generalized problems seem to become much more difficult. This chapter shows the difficulty by the failure of the incremental method, which is valid for the simple resource allocation problem, and by the so-called NP-hardness of the generalized problem.

Chapter 6 deals with a new type of resource allocation problem obtained by interchanging the objective function with the function on the left-hand side of the resource constraint in the simple resource allocation problem. This variant problem seems to be meaningful in most of practical situations where the simple resource allocation problem plays a crucial role. This chapter presents three efficient algorithms for solving the problem. It is shown that each algorithm is advantageous over the others for a certain class of the problem.

Chapter 7 deals with another type of resource allocation problem, i.e., an equipollent resource allocation problem. This problem occurs in many application fields where it is necessary

to distribute a given amount of integer resources to a given set of activities so as to minimize the unbalance of the profits arising from the resulting allocation. This chaper proposes an efficient algorithm for this problem, requiring $O(n^2 + n\log N)$ time, where n is the number of activities and N is the amount of resources.

Chapter 8 also deals with the same problem discussed in Chapter 7. This chapter proposes another efficient algorithm which requires $O(n^2 + \bar{T})$ running time, where n is the number of activities and $\bar{T}$ is the time to solve the continuous version of the problem. This algorithm is advantageous over the previous one proposed in Chapter 7 for a certain class of the problem. This chapter then applies this result to the apportionment problem. Some computational results are also reported.

Chapter 9 proposes an efficient algorithm for obtaining K best solutions of the simple resource allocation problem. It is known that K best solutions can be obtained as K shortest simple paths in the directed graph representing the multi-stage decision process obtained by the dynamic programming approach. This chapter presents a more efficient algorithm through the techniques developed in the first part of the thesis. It requires $O(T^* + \sqrt{n \log n} + K \log K)$ running time and $O(K\sqrt{n \log n} + n)$ storage space, where T^* is the computation time to obtain an optimal solution.

The final chapter gives the conclusion of the thesis, and

summarizes the results of Chapters 2 through 9. Appendix provides definitions and notations in graph theory which are frequently used in the thesis.

It is mentioned here that the material discussed in Chapter 2 is taken from the published paper [K5] and the paper under submission [K11], Chapter 3 from the published paper [K8], Chapter 4 from the published paper [K6], Chapter 5 from the published paper [K9], Chapter 6 from the published paper [K7] and the presentation [K10], Chapter 7 from the paper under submission [K14], Chapter 8 from the paper under submission [K13], Chapter 9 from the published paper [K12]. Moreover, some materials in Chapters 2, 3 and 9 are also taken from the presentation [I3].

CHAPTER 2

AN EFFICIENT ALGORITHM FOR K SHORTEST SIMPLE PATHS IN AN UNDIRECTED GRAPH

This chapter proposes an efficient algorithm for obtaining K shortest simple paths in a given undirected graph with nonnegative edge length. The idea of the algorithm is as follows. First, a subroutine is developed to efficiently find a shortest simple path among those not containing a certain prespecified path as their initial subpaths. Secondly, a sophisticated partition scheme is developed not to repeatedly generate the same path as the already obtained paths. The algorithm proceeds as follows. It partitions the set of all paths into small subsets step by step, and computes the shortest simple path, the second shortest simple path, $\cdots$, K-th shortest simple path by applying the above subroutine to the partitioned path sets. The required computation time is $O(Kn^2)$ and the storage space is $O(m + Kn)$, where n is the number of vertices and m is the number of edges.

2.1 Introduction

Given an undirected graph in which length is associated with each edge, the shortest path problem is to find the shortest length path between two designated vertices. This problem is a typical one which has been practically applied to many real

problems such as transportation problems. It has been known that some combinatorial optimization problems (e.g., minimum cost flow problem [F3], integer programming problem [N1]) can be solved by repeatedly applying the shortest path algorithm. Many algorithms solving the problem have been proposed; an $O(n^2)$ algorithm is given by Dijkstra [D1] (see also Dreyfus [D2]) and, for sparse graphs, $O(m \log n)$ algorithms are given by Johnson [J3], Tomizawa [T5], where n and m are the numbers of vertices and edges in the graph respectively.

It is natural and important (e.g., Lawler [L1]) to extend the shortest path problem to the K shortest path problem. The K shortest path problem may be classified into two types; one allows paths to have cycles and the other does not allow paths to have cycles, that is, it allows only simple paths. The first type of problem has been studied by Bellman and Kalaba [B4], Fox [F2], Hoffman and Pavley [H5], Lawler [L3], Minieka and Shier [M2], Minieka [M3], Sakarovitch [S2], Shier [S3], [S4], Weintraub [W4], Wongseelashote [W5] and others. The second type has been investigated by Clarke et al. [C4], Lawler [L1], Pollack [P1, P2, P3], Weigand [W2] and J. Yen [Y2, Y3]. Especially, Yen's algorithms have an $O(Kn^3)$ running time and can be applied to a directed graph as well as an undirected graph.

This chapter considers only undirected graphs with nonnegative edge length, and presents an efficient algorithm for the

problem of the second type. The algorithm requires $O(Kn^2)$ running time and $O(m + Kn)$ storage space. The idea of the algorithm is as follows. First, a fast subroutine is developed to find in $O(n^2)$ a shortest simple path among those not containing a certain prespecified path as their initial subpaths. Secondly, a sophisticated partition scheme is developed not to repeatedly generate the same path as the already obtained paths. The algorithm partitions the set of all simple paths into small subsets step by step, and computes the shortest simple path, the second shortest simple path, ···, K-th shortest simple path by applying the above subroutine to the partitioned path sets.

This chapter is organized as follows. Section 2.2 introduces some definitions. Section 2.3 gives an outline of the entire algorithm. Section 2.4 presents the detailed description of the algorithm. Section 2.5 proves the correctness of the algorithm and analyzes its time complexity. Section 2.6 introduces two shortest path trees corresponding to the two designated vertices, and presents an algorithm for obtaining the resulting shortest path trees. Based on these two trees, Section 2.7 presents an $O(n^2)$ subroutine for obtaining the shortest simple path among those not containing a certain prespecified path as their initial subpath.

2.2 Definitions

We begin with some definitions. Let $G = (V, E)$ be an undirected graph, where V is a set of n vertices and E is a set of m edges. A nonnegative symmetric length $d(u, v) = d(v, u)$ is given to each edge $(u, v) \in E$. The problem which this chapter deals with is to find K shortest simple paths for two designated vertices $s, t \in V$. Throughout this chapter, a path usually refers to a simple path unless a confusion occurs. The (first) shortest path, denoted π^1, is a (simple) path from s to t in G with the minimum length. By induction, the k-th shortest path π^k is defined as a (simple) path from s to t in G with the minimum length among those different from $\pi^1 - \pi^{k-1}$. Each π^k is represented by the sequence of vertices through which π^k passes.

$$\pi^k = (v^k(1), v^k(2), \ldots, v^k(q_k)) \text{ with } v^k(1) = s \text{ and } v^k(q_k) = t; \quad v^k(\alpha) \text{ denotes the } \alpha\text{-th vertex on } \pi^k. \tag{2.1}$$

$$\rho^k(\alpha) = (v^k(1), \ldots, v^k(\alpha)) \ (1 \le \alpha \le q_k): \text{ the initial portion of } \pi^k \text{ connecting } \alpha \text{ vertices.} \tag{2.2}$$

$$\sigma^k(\alpha) = (v^k(\alpha), \ldots, v^k(q_k)) \ (1 \le \alpha \le q_k): \text{ the last portion of } \pi^k \text{ connecting } q_k - \alpha + 1 \text{ vertices.} \tag{2.3}$$

$$\alpha^k = 0 \quad \text{if } k = 1,$$
$$= \max\{\alpha \mid 1 \le \alpha \le q_k, \ (\exists\, h \le k-1)(\rho^h(\alpha) = \rho^k(\alpha))\} \text{ if } k \ge 2. \tag{2.4}$$

$$f(k) = 0 \text{ if } k = 1,$$
$$= \min\{h \mid 1 \le h \le k-1,\ \rho^h(\alpha^k) = \rho^k(\alpha^k)\} \text{ if } k \ge 2. \tag{2.5}$$

$G^k(\alpha+1)$: The graph obtained from G by deleting vertices $v^k(1), v^k(2), \ldots, v^k(\alpha)$ together with edges incident with them. (2.6)

Index α^k is the largest α such that $\rho^k(\alpha)$ coincides with $\rho^h(\alpha)$ for some $h < k$; $\rho^k(\alpha^k+1)$ is not a subpath of π^h for any $h < k$. $f(k)$ is the lowest index h such that $\rho^k(\alpha^k) = \rho^h(\alpha^k)$ holds; the common initial subpath of π^k and $\pi^{f(k)}$ is $\rho^k(\alpha^k)$ $(= \rho^{f(k)}(\alpha^k))$.

The following data are also necessary not to generate the same path twice during the computation of $\pi^1 - \pi^K$.

$$W^h = \{\alpha^h + 1,\ q_h\} \cup \{\alpha^k \mid (\exists\, k > h)(f(k) = h)\} \quad (1 \le h \le k), \tag{2.7}$$

$$B^h(\alpha) = \{v^k(\alpha+1) \mid k > h,\ f(k) = h,\ \alpha^k = \alpha\} \quad (1 \le h \le k,\ \alpha^h < \alpha \le q_h). \tag{2.8}$$

Namely W^h stores the index α^k representing the vertex $v^h(\alpha^k)$ at which π^k with $h = f(k)$ deviates from π^h, and $B^h(\alpha(=\alpha^k))$ stores the vertex $v^k(\alpha^k+1)$. W^h and $B^h(\alpha)$ are computed as follows. When π^h is obtained in the algorithm, it is initially set that $W^h = \{\alpha^h + 1, q_h\}$ and $B^h(\alpha) = \phi$ for $\alpha^h + 1 \le \alpha \le q_h$. Then α^k is added to W^h and $v^k(\alpha^k+1)$ is added to $B^h(\alpha^k)$ whenever π^k satisfying $h = f(k)$ is obtained. Note that $\alpha^h + 1 \le \alpha^k \le q_k$ always holds for $h = f(k)$ as obvious from the definition (2.5) of $f(k)$.

2.3 The Outline of the Algorithm

In this section we give an outline of the entire algorithm. The following subroutine plays a key role.

FSP($\bar{G}$, $\bar{s}$, $\bar{t}$, $\bar{\pi}$) : Obtains in $O(n^2)$ time a shortest one among the simple paths from $\bar{s}$ to $\bar{t}$ not containing $\bar{\pi}$ as their initial subpaths, where $\bar{s}$, $\bar{t}$ are vertices in a graph $\bar{G}$ and $\bar{\pi}$ is an initial subpath of a shortest path from $\bar{s}$ to $\bar{t}$ in $\bar{G}$.

In particular if $\bar{\pi}$ is itself a shortest path from $\bar{s}$ to $\bar{t}$ in $\bar{G}$, FSP obtains a second shortest path. Furthermore, if $\bar{G}$ has no path from $\bar{s}$ to $\bar{t}$ satisfying the condition, FSP outputs ϕ. The details of FSP will be explained in Section 2.7.

In the main algorithm, π^1 is obtained in $O(n^2)$ time by the well known Dijkstra method [D1]. π^2 is then obtained in $O(n^2)$ time by calling FSP(G, s, t, π^1). In order to compute π^3, the set of all paths from s to t excluding π^1 and π^2 is partitioned into the following three sets (see Fig. 2.1).

(1) The set of paths from s to t which have $\rho^2(\alpha^2+1)$ as their initial subpaths but are different from π^2. The shortest path π_a among them is obtained by computing the shortest path π from $v^2(\alpha^2+1)$ to t among those different from $\sigma^2(\alpha^2+1)$ by applying FSP($G^2(\alpha^2+1)$, $v^2(\alpha^2+1)$, t, $\sigma^2(\alpha^2+1)$), and then concatenating π after $\rho^2(\alpha^2+1)$, i.e., $\pi_a = \rho^2(\alpha^2+1)\pi$. If $v^2(\alpha^2+1) = t$, this set is empty and π_a is not considered.

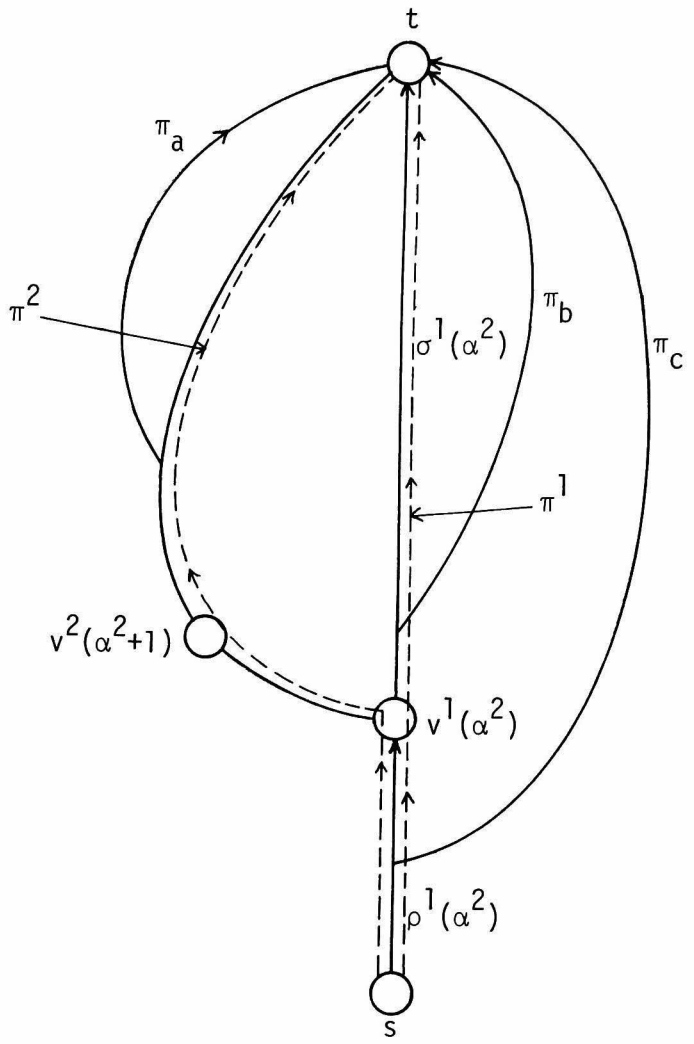

Fig. 2.1 Illustration of the relation between π^1 and π^2

(2) The set of paths from s to t which contain $\rho^1(\alpha^2)(=\rho^2(\alpha^2))$ but do not contain $\rho^2(\alpha^2+1)$ as their initial subpaths, and are different from π^1. The shortest path π_b among them is given by $\rho^1(\alpha^2)\pi'$, where π' is obtained by applying FSP(G', $v^1(\alpha^2)$, t, $\sigma^1(\alpha^2)$) and G' is $G^1(\alpha^2)$ with edge $(v^1(\alpha^2), v^2(\alpha^2+1))$ deleted.

(3) The set of paths from s to t which do not have $\rho^1(\alpha^2)$ as their initial subpaths(i.e., necessarily branch from $\rho^1(\alpha^2)$ before reaching $v^1(\alpha^2)$). The shortest path π_c among them is obtained by applying FSP(G, s, t, $\rho^1(\alpha^2)$). If $v^1(\alpha^2)=s$, this set is empty and π_c is not considered.

Now it is clear that π_a, π_b, π_c are distinct simple paths which are also different from π^1 and π^2, and that the shortest one among π_a, π_b, π_c is the third shortest path π^3. π_a, π_b and π_c not selected as π^3 are then stored in list C, which holds candidates for the next path π^k when $\pi^1 \sim \pi^{k-1}$ are already obtained. In order to obtain π^4, π^5, ..., π^K, the above procedure is repeated in the following manner.

Suppose that $W^h = \{\beta_1 (=\alpha^h+1), \beta_2, \ldots, \beta_{r_h} (=q_h)\}(1 \le h \le k-1)$ holds at the time immediately before π^k is obtained, where $\beta_1 < \beta_2 < \ldots < \beta_{r_h}$. Let the set of all paths from s to t other than $\pi^1 \sim \pi^{k-1}$ be partitioned into path sets $P^h_{k-1}(\beta_i, \beta_{i+1})$, $1 \le h \le k-1$, $1 \le i \le r_h - 1$ satisfying the following two conditions.

(i) Any path $\pi \in P^h_{k-1}(\beta_i, \beta_{i+1})$ contains $\rho^h(\beta_i)$ as its initial

subpath, and necessarily branches from π^h before reaching $v^h(\beta_{i+1})$.

(ii) No path $\pi \in P^h_{k-1}(\beta_i, \beta_{i+1})$ has $\rho^\ell(\beta_i + 1)$ as its initial subpath for any $v^\ell(\beta_i + 1) \in B^h(\beta_i)$, $h < \ell \leq k-1$ (i.e., π is different from π^ℓ such that $f(\ell) = h$ and $\alpha^\ell = \beta_i$).

Note that the path sets (1), (2) and (3) defined earlier in this section are respectively equal to $P^2_2(\alpha^2 + 1, q_2)$, $P^1_2(\alpha^2, q_1)$ and $P^1_2(1, \alpha^2)$ in this notation. Generally, as shown in Lemma 2.1, path sets $P^h_{k-1}(\beta_i, \beta_{i+1})$ are mutually disjoint and $\cup_{i,h} P^h_{k-1}(\beta_i, \beta_{i+1})$ is equal to the set of all paths from s to t other than $\pi^1 \sim \pi^{k-1}$. The shortest path in each nonempty $P^h_{k-1}(\beta_i, \beta_{i+1})$ is stored in list C and π^k is obtained as the shortest one among those stored in C. Then π^k is deleted from C and data are updated for the computation of π^{k+1}. First it is set that $W^k \leftarrow \{\alpha^k + 1, q_k\}$ and $B^k(\alpha) \leftarrow \phi$ for $\alpha^k + 1 \leq \alpha \leq q_k$, and then W^h and $B^h(\alpha)$ with $h = f(k)$ are updated as described below.

(I) If $\alpha^k \notin W^h$ (i.e., $B^h(\alpha^k) = \phi$), α^k is added to W^h and $v^k(\alpha^k + 1)$ is added to $B^h(\alpha^k)$. Let

$$\delta \leftarrow \max\{\alpha \mid \alpha^h + 1 \leq \alpha < \alpha^k, \ \alpha \in W^h\},$$
$$\gamma \leftarrow \min\{\alpha \mid \alpha^k + 1 \leq \alpha \leq q_h, \ \alpha \in W^h\}.$$

It is clear that $\pi^k \in P^h_{k-1}(\delta, \gamma)$. The set $P^h_{k-1}(\delta, \gamma) - \{\pi^k\}$ is then partitioned into three sets $P^k_k(\alpha^k + 1, q_k)$, $P^h_k(\alpha^k, \gamma)$ and $P^h_k(\delta, \alpha^k)$ (see Fig. 2.2). ($P^k_k(\alpha^k + 1, q_k) = \phi$ and is not considered

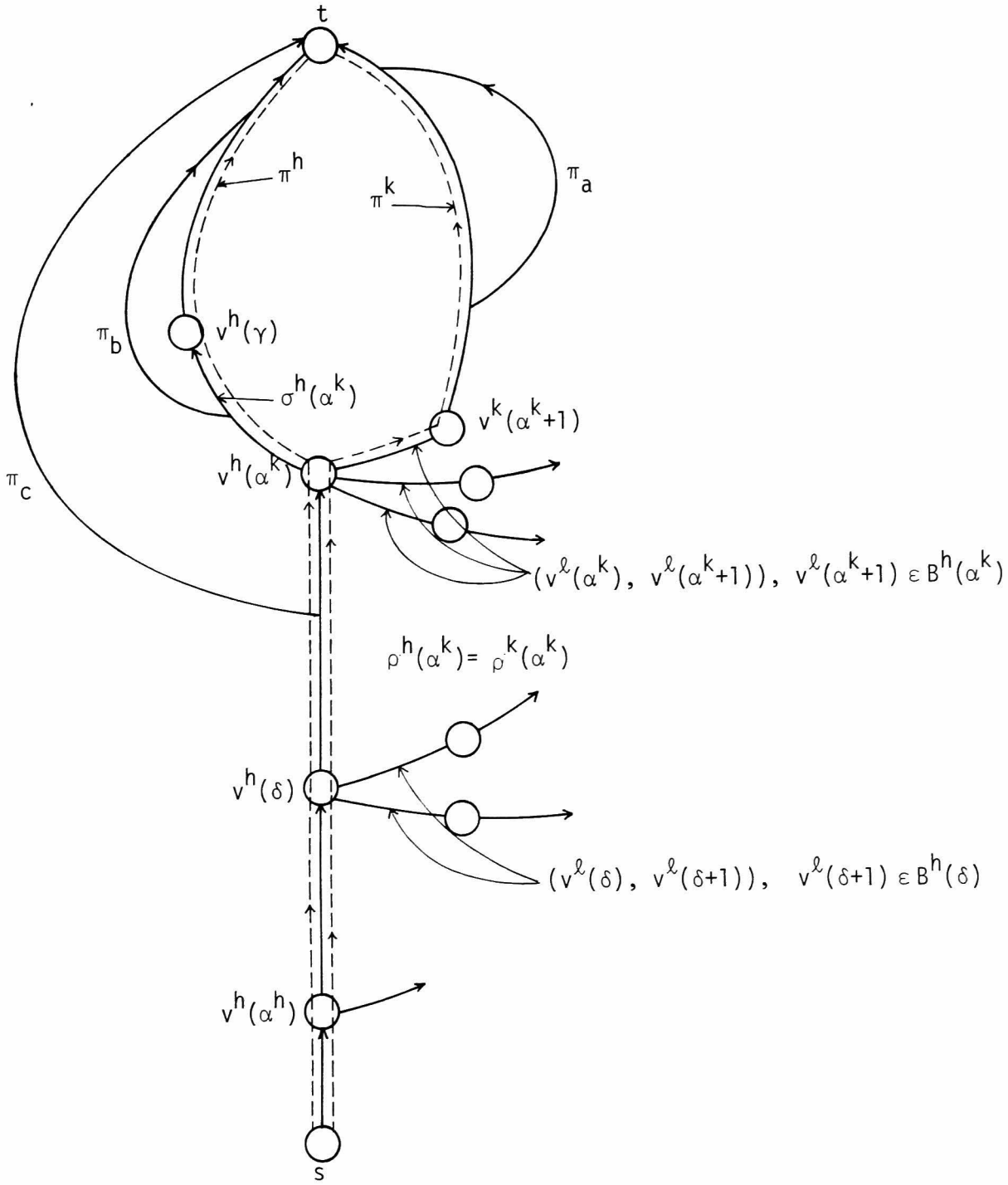

Fig. 2.2 Relative positions of edges and vertices used for computation of π^{k+1}, where $h = f(k)$.

if $\alpha^k + 1 = v^k(q_k)\ (= t)$). The shortest paths π_a, π_b, and π_c in these sets respectively are then computed and added to C. $P^h_{k-1}(\beta_i, \beta_{i+1})$'s other than $P^h_{k-1}(\delta, \gamma)$ become $P^h_k(\beta_i, \beta_{i+1})$ without change.

(II) If $\alpha^k \in W^h$, W^h does not change but $v^k(\alpha^k + 1)$ is added to $B^h(\alpha^k)$. It is clear that $\pi^k \in P^h_{k-1}(\alpha^k, \gamma)$ for the γ defined above. The set $P^h_{k-1}(\alpha^k, \gamma) - \{\pi^k\}$ is partitioned into two sets $P^k_k(\alpha^k + 1, q_k)$ and $P^h_k(\alpha^k, \gamma)$(see Fig. 2.2). The shortest path π_a and π_b in these sets are computed and added to C. $P^h_{k-1}(\beta_i, \beta_{i+1})$'s other than $P^h_{k-1}(\alpha^k, \gamma)$ become $P^h_k(\beta_i, \beta_{i+1})$ without change.

π_a, π_b and π_c defined above are computed as follows.

(a) Let π be the path obtained by applying FSP($G^k(\alpha^k + 1)$, $v^k(\alpha^k + 1)$, t, $\sigma^k(\alpha^k + 1)$). Then $\pi_a = \rho^k(\alpha^k + 1)\pi$.

(b) Let π' be the path obtained by applying FSP(G', $v^h(\alpha^k)$, t, ($v^h(\alpha^k)$, ..., $v^h(\gamma)$)), where G' is obtained from $G^h(\alpha^k)$ by deleting all edges $(v^\ell(\alpha^k), v^\ell(\alpha^k + 1))$ with $v^\ell(\alpha^k + 1) \in B^h(\alpha^k)$. Then $\pi_b = \rho^h(\alpha^k)\pi'$.

(c) Let π'' be the path obtained by applying FSP(G'', $v^h(\delta)$, t, ($v^h(\delta)$, ..., $v^h(\alpha^k)$)), where G'' is obtained from $G^h(\delta)$ by deleting all edges $(v^\ell(\delta), v^\ell(\delta + 1))$ with $v^\ell(\delta + 1) \in B^h(\delta)$. Then $\pi_c = \rho^h(\delta)\pi''$.

These π_a, π_b and π_c are computed in $O(n^2)$ steps since Subroutine FSP requires $O(n^2)$ steps as shown in Section 2.7. Repeating the above procedure for k = 2, 3, ..., K-1, paths π^3, π^4, ..., π^K are obtained. The entire computation of π^1, π^2, ..., π^K requires $O(Kn^2)$ time as proved in Theorem 2.3 of Section 2.5.

2.4 Algorithm KSP

A description of Algorithm KSP(G, s, t, K) for obtaining $\pi^1, \pi^2, \ldots, \pi^K$ in this order is now given.

We first explain some data structures. KSP uses Subroutine FSP($\bar{G}$, $\bar{s}$, $\bar{t}$, $\bar{\pi}$) as explained before, and FSP outputs path π in the following form.

$$\pi = (L, \varepsilon, q; u_1(=\bar{s}), u_2, \ldots, u_q(=\bar{t})), \tag{2.9}$$

where L is the length of π, ε denote the vertex v_ε in $\bar{\pi} = v_1, v_2, \ldots, v_p$ at which π first branches from $\bar{\pi}$, q is the number of verticer on π and $u_1, u_2, \ldots, u_q$ are the sequence of vertices representing π. A path π in list C is stored as follows:

$$\pi = (L, h, \varepsilon, q; v_1(=s), v_2, \ldots, v_q(=t)), \tag{2.10}$$

where L is the length of π and $v_1, v_2, \ldots, v_q$ represent the sequence of vertices on π. When π is chosen as π^k and deleted from C, h, ε and q work as f(k), α^k and q_k respectively. The k-th shortest path π^k is stored as follows:

$$\pi^k = (L_k, f(k), \alpha^k, q_k; v^k(1), v^k(2), \ldots, v^k(q_k)), \tag{2.11}$$

where L_k is the length of π^k and f(k), α^k, q_k and $v^k(1)$, $v^k(2)$, $\ldots$, $v^k(q_k)$ are those defined in Section 2.2. Lists W^k, $B^k(\alpha^k+1)$, $\ldots$, $B^k(q_k)$ are also associated with π^k. When π^k is computed, it is set that $W^k = \{\alpha^k+1, q_k\}$ and $B^k(\alpha) = \phi$, $\alpha = \alpha^k+1, \ldots, q_k$,

and then updated from time to time as explained after (2.8). Each index $\alpha \in W^k$ has a pointer to list $B^k(\alpha)$ if $B^k(\alpha) \neq \phi$. Graph G is represented by the adjacency list $A_G = \{A_G(u) \mid u \in V\}$, $A_G(u) = \{v \mid (u, v) \in E\}$, in which each $u \in A_G(v)$ has a pointer to $v \in A_G(u)$. Also we let each $v \in B^k(\alpha)$ have a pointer to $v \in A_G(v^k(\alpha))$.

The following is a description of Algorithm KSP written in an ALGOL-like language. We assume that G has at least one simple path from s to t (i.e., G is connected). If G does not have the k-th shortest path for $2 \leq k \leq K$, KSP terminates after generating $\pi^1, \pi^2, \ldots, \pi^{k-1}$.

```
     Procedure KSP(G, s, t, K) ; begin
     comment K ≥ 2 ;
     comment Computation of π^1 ;
1    Apply the Dijkstra algorithm to obtain π^1 = v^1(1)(=s), v^1(2),
          ..., v^1(q_1)(=t) ;
2    C ← φ; α^1 ← 0; W^1 ← {1, q_1}; f(1) ← 0;
3    for α=1 until q_1 do B^1(α) ← φ;
4    Call FSP(G, s, t, π^1) to obtain π = (L, h(=1), ε, q; u_1, ..., u_q);
5    C ← C ∪ {π};
     comment Computation of π^2, π^3, ..., π^K;
6    for k=2 until K do
         begin
7            if C = φ then stop (G does not have the k-th shortest path);
```

8 Find $\pi=(L, h, \varepsilon, q; u_1, \ldots, u_q) \in C$ with smallest L;

9 $C \leftarrow C - \{\pi\}$;

10 $\pi^k=(L_k, f(k), \alpha^k, q_k; v^k(1), \ldots, v^k(q_k))$

$\leftarrow(L, h, \varepsilon, q; u_1, \ldots, u_q)$;

11 if k=K then stop (all K shortest paths have been output);

12 $W^k \leftarrow \{\alpha^k+1, q_k\}$;

13 for $\alpha=\alpha^k+1$ until q_k do $B^k(\alpha) \leftarrow \phi$;

14 $B^h(\alpha^k) \leftarrow B^h(\alpha^k) \cup \{v^k(\alpha^k+1)\}$;

15 if $\alpha^k+1 \neq q_k$ then

begin (obtain π_a)

16 Call FSP($G^k(\alpha^k+1)$, $v^k(\alpha^k+1)$, t, $\sigma^k(\alpha^k+1)$)

to obtain $\pi=(L, \varepsilon, q; u_1, \ldots, u_q)$;

17 $\pi_a \leftarrow (L+\sum_{i=1}^{\alpha^k} d(v^k(i), v^k(i+1)), k, \alpha^k+\varepsilon,$

$\alpha^k+q; \rho^k(\alpha^k+1), u_2, \ldots, u_q)$;

18 $C \leftarrow C \cup \{\pi_a\}$;

comment π_a is the path obtained by concatenating π after $\rho^k(\alpha^k+1)$. If $\pi=\phi$ (i.e., no path is found in FSP), π_a is also ϕ and C does not change at line 18;

end

begin (obtain π_b)

19 $\gamma \leftarrow \min\{\alpha \mid \alpha^k+1 \leq \alpha \leq q_h, \alpha \in W^h\}$;

20 Let G' be the graph obtained from $G^h(\alpha^k)$ by deleting edges $(v^\ell(\alpha^k), v^\ell(\alpha^k+1))$ with $v^\ell(\alpha^k+1) \in B^h(\alpha^k)$ (note that $h = f(k)$ holds as obvious from line 10);

21 $\rho \leftarrow (v^h(\alpha^k), v^h(\alpha^k+1), \ldots, v^h(\gamma))$;

22 Call FSP(G', $v^h(\alpha^k)$, t, ρ) to obtain $\pi' = (L, \varepsilon, q; u_1, \ldots, u_q)$;

23 $\pi_b \leftarrow (L+\sum_{i=1}^{\alpha^k-1} d(v^h(i), v^h(i+1)), h, \alpha^k+\varepsilon-1, q+\alpha^k-1; \rho^h(\alpha^k), u_2, \ldots, u_q)$ (if $\pi'=\phi$ then π_b is also set to ϕ);

24 $C \leftarrow C \cup \{\pi_b\}$;

end

25 if $\alpha^k \notin W^h$ then

begin (obtain π_c)

26 $W^h \leftarrow W^h \cup \{\alpha^k\}$;

27 $\delta \leftarrow \max\{\alpha \mid \alpha^h+1 \le \alpha < \alpha^k, \alpha \in W^h\}$;

28 Let G'' be the graph obtained from $G^h(\delta)$ by deleting edges $(v^\ell(\delta), v^\ell(\delta+1))$ with $v^\ell(\delta+1) \in B^h(\delta)$;

29 $\rho \leftarrow v^h(\delta), \ldots, v^h(\alpha^k)$;

30 Call FSP(G'', $v^h(\delta)$, t, ρ) to obtain $\pi'' = (L, \varepsilon, q; u_1, \ldots, u_q)$;

31 $\pi_c \leftarrow (L+\sum_{i=1}^{\delta-1} d(v^h(i), v^h(i+1)), h, \delta+\varepsilon-1, q+\delta-1; \rho^h(\delta), u_2, \ldots, u_q)$ (if $\pi''=\phi$ then π_c is also set to ϕ);

```
32                    C ← C ∪{π_c};
                  end
              end
          end   KSP;
```

2.5 Correctness and Time Complexity of KSP

In this section we prove that KSP correctly computes $\pi^1, \pi^2, \ldots, \pi^K$ in $O(Kn^2)$ steps.

Lemma 2.1 Let $\tau(k)$ $(2 \le k \le K)$ be the instant just before π^k is obtained at line 10 of KSP. Let $W^h = \{\beta_1, \ldots, \beta_{r_h}\}$ for $h = 1, 2, \ldots, k-1$, where $\beta_1 \le \ldots \le \beta_{r_h}$, and let $P^h_{k-1}(\beta_i, \beta_{i+1})$ $(1 \le h \le k-1,\ 1 \le i \le r_h - 1)$ denote the path sets defined in Section 2.3. Then the following properties hold.

(1) $P^h_{k-1}(\beta_i, \beta_{i+1})$'s are mutually disjoint, and $\cup_{1 \le h \le k-1,\ 1 \le i \le r_h - 1} P^h_{k-1}(\beta_i, \beta_{i+1})$ is equal to the set of all paths from s to t in G excluding $\pi^1 \sim \pi^{k-1}$.

(2) For each nonempty $P^h_{k-1}(\beta_i, \beta_{i+1})$, the shortest path in the set is stored in C at $\tau(k)$.

(3) Each path stored in C at $\tau(k)$ is the shortest path in some $P^h_{k-1}(\beta_i, \beta_{i+1})$.

Proof. Proof is done by induction on k.

[Basis]: k=2. At $\tau(2)$, $W^1 = \{1, q_1\}$ and $B^1(\alpha) = \phi$ $(1 \le \alpha \le q_1)$ hold as easily seen from lines 2 and 3 of KSP. We have only $P^1_1(1, q_1)$ which is equal to the set of all paths from s to t excluding π^1. Thus (1) is obvious. The shortest path π is $P^1_1(1, q_1)$(i.e., the second shortest path in G) is computed at line 4 (by the correctness of Subroutine FSP that will be shown in Section 2.7) and stored in C at line 5; C contains only π. Hence (2) and (3) are true.

[Induction step] (see Fig. 2.2) We assume that (1), (2) and (3) hold at $\tau(k)$, and prove them for $\tau(k+1)$. Assume further that $\alpha^k+1 \neq q_k$ (see line 15) and $\alpha^k \notin W^h$ (see line 25) since other cases can be simlarly treated. First note that $\pi^k \in P^h_{k-1}(\delta, \gamma)$ holds at $\tau(k)$, where h is the one obtained at line 8 (i.e., $h = f(k)$ holds) and γ and δ are those obtained at lines 19 and 27 respectively between $\tau(k)$ and $\tau(k+1)$. The definition of W^k at line 12 and addition of $\{\alpha^k\}$ to W^h at line 26 then imply the introduction of three path sets $P^k_k(\alpha^k+1, q_k)$, $P^h_k(\alpha^k, \gamma)$ and $P^h_k(\delta, \alpha^k)$ (see the discussion (I) of Section 2.3). Any path $\pi \in P^h_k(\alpha^k, \gamma)$ has $\rho^h(\alpha^k)$ as its initial subpath while no path $\pi' \in P^h_k(\delta, \alpha^k)$ does (see the definition (i) of Section 2.3). In addition, no path $\pi \in P^h_k(\delta, \alpha^k) \cup P^h_k(\alpha^k, \gamma)$ has $\rho^k(\alpha^k+1)$ as its initial subpath while $\pi' \in P^k_k(\alpha^k+1, q_k)$ does. Thus these three subsets are mutually disjoint. It is also easy to see that their union is equal to $P^h_{k-1}(\delta, \gamma) - \{\pi^k\}$. This proves (1). Next note that π_a, π_b and π_c computed at lines 17, 23 and 31 are respectively the shortest paths in these three path sets, as easily proved from the correctness of FSP (Theorem 2.6). π_a, π_b and π_c are added to C at lines 18, 24 and 32 respectively (if the corresponding sets are empty, π_a, π_b and π_c are not added to C, as commented at lines 18, 23 and 31). Since other path sets $P^\ell_{k-1}(\beta_i, \beta_{i+1})$ do not change and π^k is deleted from C at line 9, (2) and (3) are proved. □

Theorem 2.2 Algorithm KSP correctly obtains π^1, π^2, ..., π^K and halts at line 11 if G has at least K simple paths from s to t. If G has only k-1 < K simple paths from s to t, KSP halts at line 7 after obtaining π^1, π^2,..., π^{k-1}.

Proof. Assume that G has π^1, π^2, ..., π^K. Proof is done by induction on k.

[Basis] π^1 is correctly obtained. This follows from the correctness of Dijkstra's algorithm.

[Induction step] Assuming that $\pi^1 \sim \pi^{k-1}$ are correctly obtained, it is proved that π^k obtained by KSP is in fact the k-th shortest simple path.

First note that π^k obtained at line 10 is different from $\pi^1 \sim \pi^{k-1}$ by (1) and (3) of Lemma 2.1. Now let π^* be the shortest path from s to t in G, which is different from π^1, π^2, ..., π^{k-1}. π^* belongs to some $P^h_{k-1}(\beta_i, \beta_{i+1})$ by (1) of Lemma 2.1, and the shortest path in $P^h_k(\beta_i, \beta_{i+1})$ is stored in C by (2) of Lemma 2.1. Since the shortest one in C is selected as π^k at line 10 of KSP, the length of π^k is not greater than (actually equal to) the length of π^* (although $\pi^k \neq \pi^*$ may possibly hold). Thus π^k is correctly computed.

Finally if G does not have π^k, C becomes empty after generating $\pi^1 \sim \pi^{k-1}$, as obvious from (1) ~(3) of Lemma 2.1. Thus KSP halts at line 7 after outputting $\pi^1 \sim \pi^{k-1}$. □

Theorm 2.3 Algorithm KSP computes π^1, π^2, ..., π^K in $O(Kn^2)$ time and $O(m+Kn)$ space, where n is the number of vertices in G.

Proof. We list up only lines in KSP which require at least $O(n)$ steps. Line 1 requires $O(n^2)$ steps as is well known. FSP called at lines 4, 16, 22 and 30 requires $O(n^2)$ steps as will be proved in Section 2.7. The manipulation of list C at lines 5, 8, 9, 18, 24 and 32 are respectively done in $O(\log K)$ steps if an appropriate data structure such as heap is used to represent C (e.g., [K16]), since the number of paths stored in C is $O(K)$ (at most $2K$). Computation of π^k, π_a, π_b and π_c at lines 10, 17, 23 and 31 is done in $O(n)$ time since a simple path has at most n vertices. The manipulations of W^k, $B^k(\alpha)$, W^h and $B^h(\alpha^k)$ at lines 2, 3, 12, 13, 14 and 26 are done in $O(n)$ steps by $|W^k| \leq n$ and $|B^k(\alpha)| \leq n$. Graphs $G^k(\alpha^k+1)$, G', G'' used at lines 16, 20 and 28 can be respectively constructed in $O(n)$ steps as easily proved (recall the data structure explained at the beginning of Section 2.4). Similarly γ and δ at lines 19 and 27 are obtained in $O(n)$ steps, and lines 21 and 29 are also $O(n)$ computation. Note that this time estimation is valid even if other minor operations such as setting up the pointers explained before the description of KSP are included.

Since lines 6-32 are repeated K-1 times, $O(Kn^2+K \log K)$ steps are required in total. However, the number of simple paths from

s to t is less than 2^m (m is the number of edges), i.e., $K < 2^m$. Thus $O(Kn^2 + K \log K) = O(Kn^2)$ by $m < n^2$.

Finally we briefly consider the space requirement. The required space is dominated by O(m) to store G and O(Kn) to store C and $B^h(\alpha^k)$'s (at most Kn $B^h(\alpha^k)$ are prepared and the size of at most one $B^h(\alpha^k)$ increases by one when the loop of lines 6-32 is executed once). (W^h)'s require O(K) space because initially they require O(K) and at most three vertices are added to them during one execution of lines 6-32. Others necessary as working space are obviously O(n). Thus O(m + Kn) space is required in total. □

2.6 Shortest Path Trees T(s) and T(t)

Before explaining FSP in the next section, we introduce two trees T(s) and T(t) of shortest paths. These are computed by subroutine DIJKSTRA in $O(n^2)$ steps.

Denote the unique path from u to v in a tree T or on a path π by $u \xrightarrow[T]{*} v$ or $u \xrightarrow[\pi]{*} v$ respectively. Define tree T(s) (resp. T(t)) for $\pi^1 = (v^1(1)\ (=s),\ v^1(2),\ \ldots,\ v^1(q_1)\ (=t))$ as follows.

(1) For each vertex $v \in V$, $s \xrightarrow[T(s)]{*} v$ (resp. $t \xrightarrow[T(t)]{*} v$) is a shortest path from s (resp. t) to v. If there is no path from s (resp. t) to v, v is not a vertex in T(s) (resp. T(t)).

(2) For each vertex $v^1(\alpha)$ on π^1 with $1 \le \alpha \le q_1$, $s \xrightarrow[T(s)]{*} v^1(\alpha)$ (resp. $v^1(\alpha) \xrightarrow[T(t)]{*} t$) is equal to $\rho^1(\alpha)$ (resp. $\sigma^1(\alpha)$).

(3) For each vertex $u \in V$, let $v^1(\xi(u))$ (resp. $v^1(\zeta(u))$) be the vertex at which $s \xrightarrow[T(s)]{*} u$ (resp. $t \xrightarrow[T(t)]{*} u$) branches from π^1. Then the following inequality always holds.

$$\xi(u) \le \zeta(u). \qquad (2.12)$$

Tree T(s) is computed by DIJKSTRA(G, s, t, π^1) and Tree T(t) is computed by DIJKSTRA(G, t, s, $\overleftarrow{\pi}^1$), where $\overleftarrow{\pi}^1 = v^1(q_1), v^1(q_1 - 1), \ldots, v^1(1)$. The algorithm DIJKSTRA(G, s, t, π^1) is basically the same as the Dijkstra method [D1, D2] for obtaining the shortest paths from s to all the other vetices, except that the following modification is introduced to guarantee (2) and condition (2.12). When more than one shortest path to a vertex $u \in V$ (with the same

length) is found during computation by the Dijkstra method, (a) if $u = v^1(\alpha)$ for some $1 \le \alpha \le q_1$ then $\rho^1(\alpha)$ is stored as the shortest path to u, and (b) if u is not on π^1 then the path which branches from π^1 at the vertex closest to s is stored as the shortest path to u.

The detailed description of DIJKSTRA is omitted since the above modification is straightforward.

For each vertex $u \in V$, let $\pi_s(u)$ denote the path $s \xrightarrow[T(s)]{*} u$. The following data are output for all $u \in V$ by DIJKSTRA as a result of computing T(s): The length $L_s(u)$ of $\pi_s(u)$, the vertex $F_s(u)$ immediately preceding u on $\pi_s(u)$, the index $\xi(u)$ representing $v^1(\xi(u))$ at which $\pi_s(u)$ first branches from π^1 and the number of vertices $q_s(u)$ on $\pi_s(u)$. Similarly, $L_t(u)$, $F_t(u)$, $\zeta(u)$ and $q_t(u)$ are output for $\pi_t(u) = t \xrightarrow[T(t)]{*} u$. Note that $\xi(u) = \zeta(u) = \alpha$ holds if $u = v^1(\alpha)$ $(1 \le \alpha \le q_1)$.

Lemma 2.4 Subroutine DIJKSTRA correctly computes T(s) and T(t) satisfying the above conditions (1), (2) and (3) in $O(n^2)$ steps. □

2.7. Subroutine FSP

In this section we present FSP(G, s, t, $\rho^1(\alpha)$) which computes in $O(n^2)$ steps the shortest simple path from s to t not containing $\rho^1(\alpha)$ as its initial subpath. The following lemma gives a key property for the construction of FSP.

Lemma 2.5 Let T(s) and T(t) be the shortest path trees as defined in Section 2.6. If G has simple paths from s to t not containing $\rho^1(\alpha)$, there is a shortest one among them which is either of type 1 or of type 2.

Type 1: $s \xrightarrow[T(s)]{*} u \xrightarrow[T(t)]{*} t$, where $\xi(u) < \alpha$.

Type 2: $s \xrightarrow[T(s)]{*} u \longrightarrow v \xrightarrow[T(t)]{*} t$, where $(u, v) \in E$ is neither in T(s) nor in T(t), and $\xi(u) < \alpha$.

Proof. Let π be a shortest path from s to t not contsining $\rho^1(\alpha)$, and let $v^1(\beta)$ be the node at which π first branches from π^1 (then $\beta < \alpha$). Let (u, v) be the first edge on π which is not in T(s). Then $\beta = \xi(u)$ and hence $s \xrightarrow[\pi]{*} u = s \xrightarrow[T(s)]{*} u$. Assume that $v \xrightarrow[\pi]{*} t$ does not coincide with $v \xrightarrow[T(t)]{*} t$ since otherwise the lemma is proved.

Case 1: $\beta < \zeta(v)$.

Subcase 1A: $v^1(\beta) \xrightarrow[T(s)]{*} u$ and $v \xrightarrow[T(t)]{*} v^1(\zeta(v))$ do not have any vertex in common. Let $\pi' = s \xrightarrow[T(s)]{*} u \longrightarrow v \xrightarrow[T(t)]{*} t$ (see Fig. 2.3). Then π' is of either type 1 or type 2, and is simple. Since $v \xrightarrow[T(t)]{*} t$ is not longer than $v \xrightarrow[\pi]{*} t$ by definition of T(t), the length of π' is not greater than that of π. Thus π' is a shortest

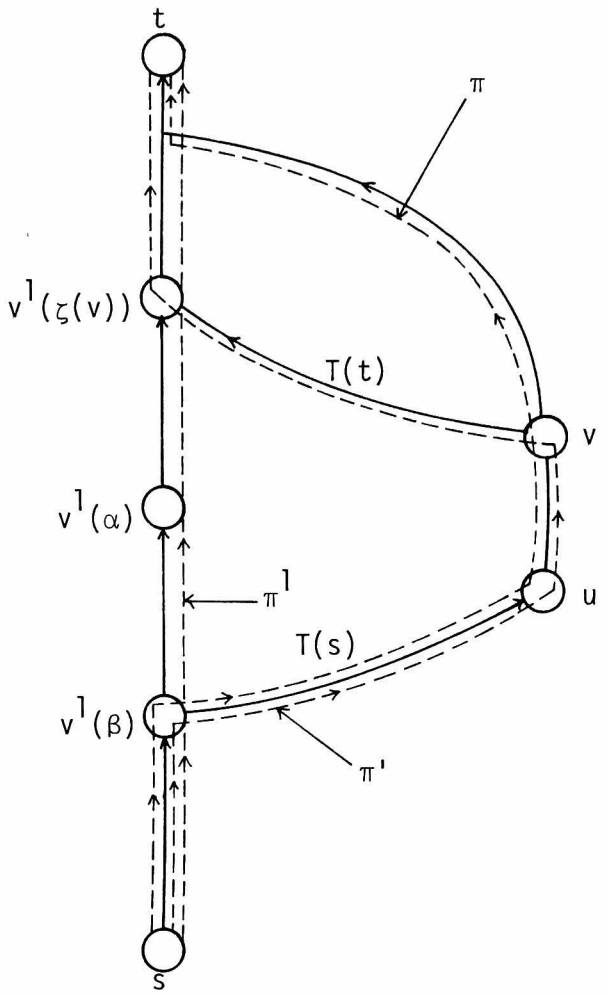

Fig. 2.3 Illustration of π and π' in Subcase 1A of the proof of Lemma 2.5.

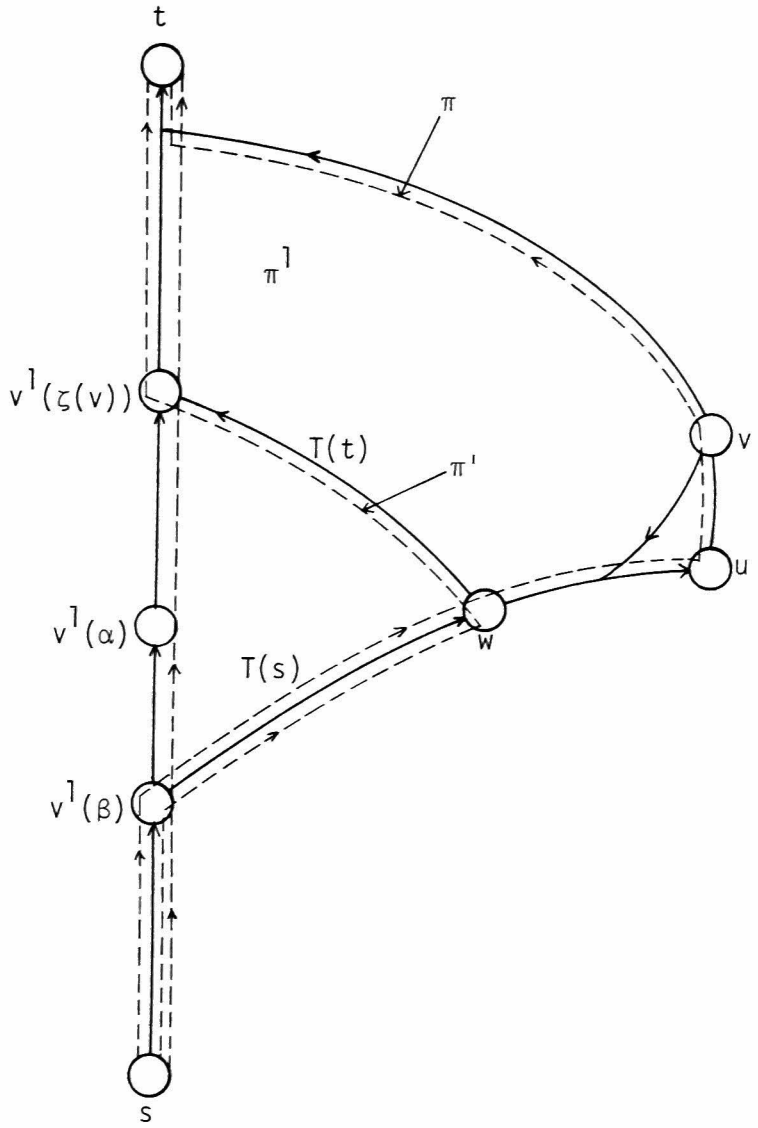

Fig. 2.4 Illustration of π and π' in Subcase 1B of the proof of Lemma 2.5.

simple path not containing $\rho^1(\alpha)$ which is of type 1 or type 2.

Subcase 1B: $v^1(\beta) \xrightarrow[T(s)]{*} u$ and $v \xrightarrow[T(t)]{*} v^1(\zeta(v))$ have some vertices in common. Among the common vertices, let w be the one nearest to $v^1(\beta)$ on $v^1(\beta) \xrightarrow[T(s)]{*} u$. Then the path $\pi' = s \xrightarrow[T(s)]{*} w \xrightarrow[T(t)]{*} t$, shown in Fig. 2.4, is simple and of type 1. Since $w \xrightarrow[T(t)]{*} t$ is not longer than $v \xrightarrow[\pi]{*} t$ by definition, π' is not longer than π.

Case 2: $\zeta(v) \leq \beta$. Let w (possibly $w = v$) be the vertex on $v^1(\xi(v)) \xrightarrow[T(s)]{*} v$ which is closest to $v^1(\xi(v))$ among those belonging to both $v^1(\xi(v)) \xrightarrow[T(s)]{*} v$ and $v \xrightarrow[\pi]{*} t$ (see Fig. 2.5). Then $\pi' = s \xrightarrow[T(s)]{*} w \xrightarrow[\pi]{*} t$ is simple and w is not on π^1 (i.e., $\pi' \neq \pi^1$) because $v \xrightarrow[\pi]{*} t$ does not have a common vertex with $s \xrightarrow[T(s)]{*} v^1(\xi(v)) = s \xrightarrow[\pi^1]{*} v^1(\xi(v))$ by $\xi(v) \leq \zeta(v) \leq \beta$ (see (2.12)) and by the simplicity of π. Moreover π' is not longer than π, since $s \xrightarrow[T(s)]{*} w$ is not longer than $s \xrightarrow[\pi]{*} w$ by definition of T(s). It is also obvious that π' does not contain $\rho^1(\alpha)$. Consequently, the lemma is proved if π' is of type 1 or type 2. Otherwise we can apply the same argument again after replacing π' by π. The latter case does not occur indefinitely, however, since the number of edges on path $w \xrightarrow[\pi]{*} t$ strictly decreases at each iteration. □

Subroutine FSP first computes T(s) and T(t) by calling DIJKSTRA. Then FSP generates all the paths of type 1 and 2, and finds the shortest one. For that, one of the following routine

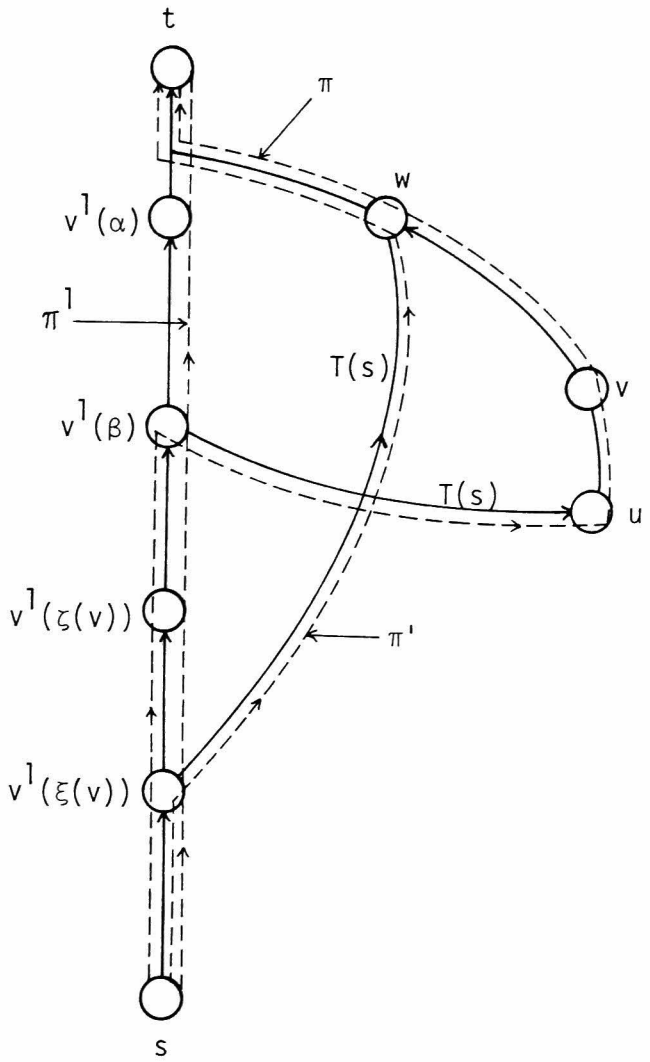

Fig. 2.5 Illustration of π and π' in Case 2 of the proof of Lemma 2.5.

is executed for each $u \in V$ with $\xi(u) < \alpha$ depending upon whether $\xi(u) = \zeta(u)$ or $\xi(u) < \zeta(u)$.

(i) If $\xi(u) = \zeta(u)$, the path of type 1, $s \xrightarrow[T(s)]{*} u \xrightarrow[T(t)]{*} t$, can be ignored since this is either equal to π^1 (i.e., contain $\rho^1(\alpha)$) or not simple. However, for each edge $(u, v) \in E$ which is neither in T(s) nor in T(t), and satisfies $\xi(u) < \zeta(v)$, the path of type 2 $s \xrightarrow[T(s)]{*} u \longrightarrow v \xrightarrow[T(t)]{*} t$ is generated. (Condition $\xi(u) < \zeta(v)$ is necessary since otherwise the generated path is not simple.) We remark here that an edge (u, v) satisfies the above conditions if and only if $\xi(u) < \zeta(v)$ and $v \in A_G(u) - S(u)$, where $S(u) = \{v \mid F_s(v) = u\}$ is the set of sons of u in T(s) viewed from s. The only-if-part is obvious since any (u, v) with $v \in S(u)$ is in T(s). To prove the if-part assume that such an edge (u, v) is in T(t). Since (u, v) is not on π^1 (by $v \notin S(u)$), $\zeta(u) = \zeta(v)$ holds by definition of T(t), and hence $\xi(u) = \zeta(u) = \zeta(v)$, a contradition.

(ii) If $\xi(u) < \zeta(u)$, only the path of type 1, $s \xrightarrow[T(s)]{*} u \xrightarrow[T(t)]{*} t$, is generated since $u \longrightarrow v \xrightarrow[T(t)]{*} t$ is not shorter than $u \xrightarrow[T(t)]{*} t$ for any $(u, v) \in E$ by definition of T(t).

The generation of the above paths is done by starting with $u = s$ and then proceeding to other vertices in such a way that vertices u closer to s in T(s) are checked earlier. FSP uses a list H to keep the current shortest path π generated in (i) or (ii), and uses a list L to store its length. It is sufficient to keep only vertex u or edge (u, v) to identify π, depending upon whether π is of type

1 or type 2.

In the above computation, paths generated in (i) and (ii) are not all simple. However it is guaranteed that a path stored in H is always simple (see Theorem 2.6). Upon termination, therefore, list H indicates the desired shortest simple path π not containing $\rho^1(\alpha)$. π is then constructed and output in the following form,

$$\pi = (L, \varepsilon, q; v_1, v_2, \ldots, v_q),$$

where L denotes the length of π, $v^1(\varepsilon)$ denotes the vertex at which π branches from π^1, q denotes the number of vertices on π and $(v_1, v_2, \ldots, v_q)$ is the sequence of vertices on π.

The search of vertices u in the manner as described above is carried out by recursively calling SEP(w), which is a subroutine to obtain the shortest path among those generated by (i) and (ii) for all descendants of w on T(s) (i.e., the vertices in the locations further than w when viewed from s).

Procedure FSP(G = (V, E), s, t, $\rho^1(\alpha)$); begin

1 Call DIJKSTRA(G, s, t, π^1) to obtain $L_s(u)$, $F_s(u)$, $q_s(u)$ and $\xi(u)$ for $u \in V$ (these are defined in Section 2.6);

2 Call DIJKSTRA(G, t, s, $\overleftarrow{\pi}^1$) to obtain $L_t(u)$, $F_t(u)$, $q_t(u)$ and $\zeta(u)$ for $u \in V$;

3 for $u \in V$ do $S(u) \leftarrow \phi$;

4 for $u \in V$ do $S(F_s(u)) \leftarrow S(F_s(u)) \cup \{u\}$;

comment S(u) stores the set of sons of u (viewed from s) in T(s);

5 $u \leftarrow s$; $H \leftarrow \phi$; $L \leftarrow \infty$;

6 Call SEP(u, $\rho^1(\alpha)$) to obtain H and L;

7 if $H = (u, v)$ then $\pi \leftarrow (L, \xi(u), q_s(u) + q_t(v); s \xrightarrow[T(s)]{*} u \longrightarrow v \xrightarrow[T(s)]{*} t)$;

8 if $H = u$ then $\pi \leftarrow (L, \xi(u), q_s(u) + q_t(u) - 1; s \xrightarrow[T(s)]{*} u \xrightarrow[T(t)]{*} t)$;

9 if $H = \phi$ then $\pi \leftarrow \phi$;

comment In the first two cases, π gives the shortest path with length L not containing $\rho^1(\alpha)$ as its initial subpath, while there is no such path in the last case.

return

end FSP;

Procedure SEP(u, $\rho^1(\alpha)$); begin

comment G, $L_s(u)$, $L_t(u)$, S(u), $\xi(u)$ and $\zeta(u)$ are those defined in FSP;

1 if $\xi(u) = \zeta(u)$ then

2 for $v \in A_G(u) - S(u)$ with $\xi(u) < \zeta(v)$ do

3 $D \leftarrow L_s(u) + d(u, v) + L_t(v)$;

4 if $D < L$ then $L \leftarrow D$, $H \leftarrow (u, v)$;

5 for $v \in S(u)$ with $\xi(v) < \alpha$ do Call SEP(v, $\rho^1(\alpha)$);

6 if $\xi(u) < \zeta(v)$ then

7 $D \leftarrow L_s(u) + L_t(u)$;

8 if $D < L$ then $L \leftarrow D$, $H \leftarrow u$;

9 for $v \in S(u)$ with $\xi(v) < \alpha$ do Call SEP(v, $\rho^1(\alpha)$);

return

end SEP;

Theorem 2.6 Algorithm FSP(G, s, t, $\rho^1(\alpha)$) correctly computes a shortest simple path from s to t in G not containing $\rho^1(\alpha)$ as its initial subpath if one exists. If G does not have such a path, FSP outputs $\pi = \phi$. In either case, FSP requires $O(n^2)$ time.

Proof. [Correctness] First assume that G has paths satisfying the above conditions. We shall show that the path π obtained at the end of FSP is the shortest one among such paths. By Lemma 2.5 and the discussion given prior to FSP, we only need to show that π obtained by FSP is a simple path. (Note that SEP does not explicitly check whether a path stored in H is simple.) Now assume that the final contents of H and L are set when SEP(u, $\rho^1(\alpha)$) is called for a vertex $u \in V$. Two cases are possible.

Case 1: The final content of H was set at line 4 of SEP(u, $\rho^1(\alpha)$), i.e., $\pi = s \xrightarrow[T(s)]{*} u \longrightarrow v \xrightarrow[T(t)]{*} t$ for some $(u, v) \in E$. Note that $(u, v) \notin T(s)$ and $(u, v) \notin T(t)$ as explained in (i) before the algorithm description. If π is not simple, let y be the vertex on $s \xrightarrow[T(s)]{*} u$ closest to s such that y is on both $s \xrightarrow[T(s)]{*} u$ and $v \xrightarrow[T(t)]{*} t$ (see Fig. 2.6). Then the path $\pi' = s \xrightarrow[T(s)]{*} y \xrightarrow[T(t)]{*} t$ is simple, not longer than π, and does not contain $\rho^1(\alpha)$ as its

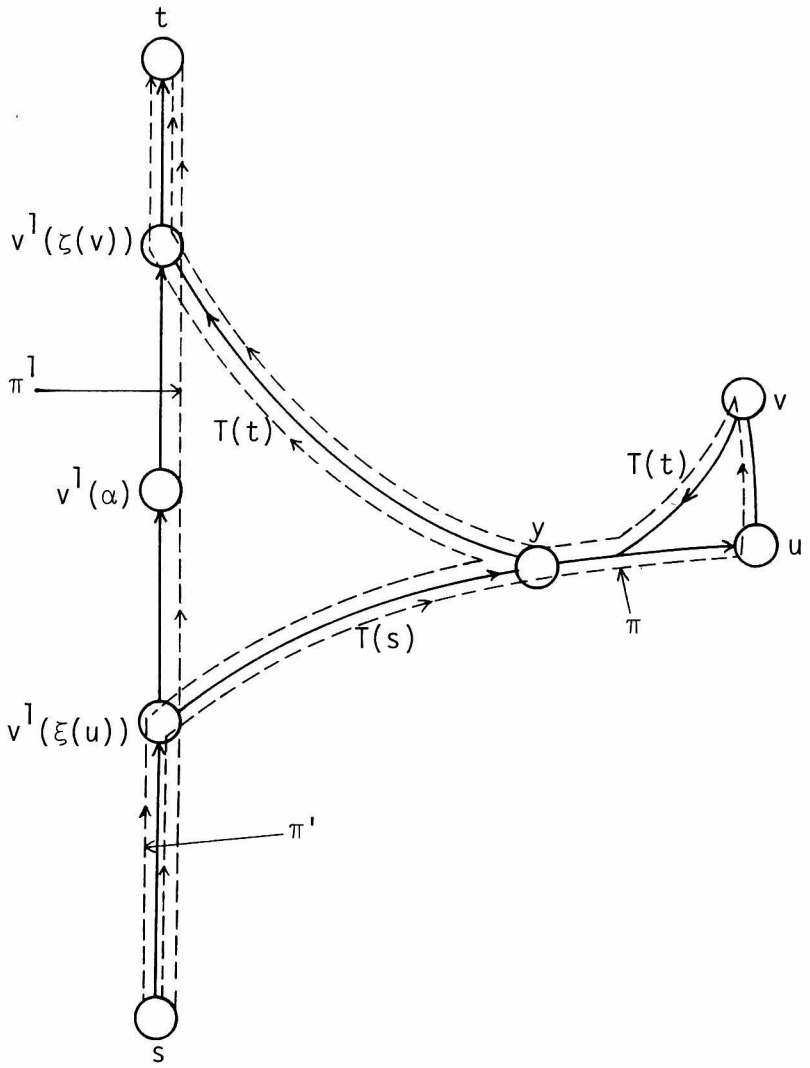

Fig. 2.6 Illustration of π and π' in Case 1 of the proof of Theorem 2.6.

initial subpath (since y is not on π^1 by the condition $\xi(u) < \zeta(v)$ of line 2). SEP(y, $\rho^1(\alpha)$) must have been executed before SEP(u, $\rho^1(\alpha)$) since y is a proper ancestor of u in T(s). Thus (u, v) can not be stored in H since π' or a shorter path is already in H. This is a contradiction.

Case 2: The final content of H was set at line 8 of SEP(u, $\rho'(\alpha)$) i.e., $\pi = s \xrightarrow[T(s)]{*} u \xrightarrow[T(t)]{*} t$. If π is not simple, let y be the vertex on $s \xrightarrow[T(s)]{*} u$ closest to s such that y is on both $s \xrightarrow[T(s)]{*} u$ and $v \xrightarrow[T(t)]{*} t$, and apply the above argument to π and $\pi' = s \xrightarrow[T(s)]{*} y \xrightarrow[T(s)]{*} t$. We also have a contradiction.

Next if G does not have a simple path not containing $\rho^1(\alpha)$ as its initial path, the above proof also shows that none is set to H during the recursive calls of SEP. Thus H remains to be ϕ and $\pi = \phi$ is output at line 9 of FSP.

[Computational Complexity] Lines 1 and 2 of FSP require $O(n^2)$ steps by Lemma 2.4. Lines 3 and 4 of FSP require O(n) steps. Line 5 of FSP requires constant time. Lines 7-9 of FSP require O(n) steps to restore the vertex sequence of π by using F_s and F_t.

Finally we consider line 6, i.e., computation of SEP. Line 1 of SEP(u, $\rho^1(\alpha)$) requires constant time. Letting set $A_G(u) - S(u)$ of line 2 be constructed when SEP(u, $\rho^1(\alpha)$) is called (this is done in $O(|A_G(u)|) \leq O(n)$ steps since $|A_G(u)| \leq n-1$ and $|S(u)| \leq n-1$), each vertex $v \in A_G(u) - S(u)$ of line 2 is generated in constant time. Therefore lines 2-4 require $O(|A_G(u) - S(u)|) \leq O(A_G(u)) \leq O(n)$

time. Lines 6-8 of SEP require constant time. Thus one execution of SEP(u, $\rho^1(\alpha)$) repuires $0(|A_G(u)|)$ time plus the time required at line 5 or 9 to call SEP(v, $\rho^1(\alpha)$) for $v \in S(u)$ with $\xi(v) < \alpha$. Since SEP(u, $\rho^1(\alpha)$) is called once for each $u \in V$, the required total time is $\sum_{u \in V} 0(|A_G(u)|) = 0(m)$ $(\leq 0(n^2))$. □

2.8 Conclusion

This chapter has proposed an $O(Kn^2)$ algorithm for obtaining K shortest simple paths in a given undirected graph with non-negative edge length. This method is superior to the existing methods, the best of which requires $O(Kn^3)$ running time. The efficiency improvement has been attained by developing a subroutine to find in $O(n^2)$ time a shortest simple path among those not containing a certain prespecified path as their initial subpaths, and a sophisticated partition scheme not to generate the same path repeatedly.

It is noted that, when the graph is sparse, the shortest path can be obtained in time less than $O(n^2)$, e.g., O(mlogn) (see [J3], [T5]). This fact can be incorporated to our algorithm as follows. Supposing that shortest paths from s to other vertices are obtained in c(n, m) ($\geq$ O(m)) time, KSP requires only O(K c(n, m)) time when line 1 of KSP and lines 1,2 of FSP are executed with such an algorithm, since the rest of KSP is computed in O(Km) time and the rest of FSP is computed in O(m) as shown in the proofs of Theorem 2.3 and Theorem 2.6 respectively.

We leave it for the future research to consider graphs possibly with negative length edges or to extend our approach to directed graphs.

CHAPTER 3

AN EFFICIENT ALGORITHM FOR FINDING K MINIMUM SPANNING TREES

This chapter presents an efficient algorithm for finding K minimum spanning trees in a given undirected graph in which weight is associated with each edge. The algorithm is based on three subroutines. The first two subroutines are used to obtain the second minimum spanning tree in $O(\min (n^2, m\ \alpha(m,n)))$ time. The third one obtains the k-th minimum spanning tree in O(m) time for the given h-th minimum spanning trees for $h = 1,2,\cdots, k-1$. Development of the sophisticated enumeration scheme together with the above three subroutines makes it possible to yield an efficient algorithm. The algorithm partitions the set of all spanning trees into small subsets step by step, and computes the minimum spanning tree, the second minimum spanning tree, $\cdots$, the K-th minimum spanning tree by applying the three subroutines to the partitioned subsets. The required computation time is $O(Km + \min (n^2, m\log\log n))$ and the storage space is O(k + m), where n and m are the numbers of vertices and edges respectively.

3.1 Introduction

Given an undirected graph in which weight is associated with each edge, the minimum spanning tree problem is to find

the spanning tree with the minimum weight. In addition to the shortest path problem discussed in Chapter 2, the minimum spanning tree problem is also one of the most important graph optimization problems. This problem has been practically applied to some real problems such as design of communication networks and cluster analysis etc. It has been known [H3, H4] that some combinatorial optimization problems (e.g., travelling salesman problem) can be reduced to the minimum spanning tree problem if some of the constraints are relaxed. Having the solution of the relaxed problem, such combinatorial optimization problems can be solved by applying the branch-and-bound method. In these respects, the minimum spanning tree problem itself is not only interesting, but also plays an important role in the field of mathematical programming.

Algorithms for solving the minimum spanning tree problem have been proposed by many authors including Kruskal [K18], Dijkstra [D1], Prim [P5], Yao [Y1], Cheriton and Tarjan [C3]. Especially [Y1] and [C3] require O(m loglog n) time, where n and m are the numbers of vertices and edges.

This chapter deals with the K minimum spanning tree problem which is to find the minimum spanning tree, the second minimum spanning tree, ···, K-th minimum spanning tree. This problem arises in many situations where, not only one minimum spanning tree is needed but also some alternative solutions (i.e., K minimum

spanning trees) are required. In addition, in case it is necessary to solve certain types of combinatorial optimization problems (e.g., travelling salesman problem) which are reduced to the minimum spanning tree problem as their relaxation problems, an optimal solution should be found among K minimum spanning trees.

Algorithms for finding K minimum spanning trees have been studied by Burns and Haff [B8], Camerini, Frata and Maffioli [C1] and Gabow [G1]. Gabow's algorithm requires $O(Km\ \alpha(m, n) + m \log n)$ time and $O(K + m)$ space, where α is Tarjan's inverse of Ackermann function [T2] and is very slowly growing.

This chapter proposes an algorithm with $O(Km + \min(m \log \log n, n^2))$ time and $O(K + m)$ space. This is slightly faster than Gabow's algorithm and the storage space is in the same order. The proposed algorithm consists of three subroutines. The first two subroutines obtain the second minimum spanning tree in $O(n^2)$ and in $O(m\ \alpha(m,n))$ time respectively. Depending upon whether $n^2 < m\ \alpha(m, n)$ or not, the algorithm chooses the faster one. The third one obtains the k-th minimum spanning tree in $O(m)$ time for the given h-th minimum spanning trees for h=1,2, ..., k-1. Incorporating the above three subroutines into the sophisticated partition scheme (slightly different from the one proposed in Gabow [G1]), the algorithm systematically computes the minimum spanning tree, the second minimum spanning tree, $\cdots$, the K-th minimum spanning tree.

This chapter is organized as follows. Section 3.2 describes definitions and a property of edge exchanges. Sections 3.3 and 3.4 give the outline of the entire algorithm and its detailed description respectively. Section 3.5 analyzes time and space requirement, i.e., $O(Km + \min(n^2, m \log\log n))$ time and $O(Km)$ space. Section 3.6 and 3.7 describe subroutines computing edge exchanges to generate the second and the k-th minimum spanning trees for any k, respectively. Section 3.8 improves the space required for the algorithm presented in Sections 3.3 and 3.4, and gives an improved algorithm with $O(K + m)$ space.

3.2 Definitions

Let $G = (V, E)$ be an undirected connected graph with no parallel edges, where V is a set of n vertices and E is a set of m edges. The weight $w(e)$ is associated with each edge $e \in E$. The weight of a spanning tree T (viewed as a set of edges) is defined by $w(T) = \sum_{e \in T} w(e)$. The k-th minimum spanning tree T^k is defined recursively as follows.

(1) T^1 is the minimum spanning tree .

(2) T^k $(k \geq 2)$ is a spanning tree with the minimum weight among those different from T^1, T^2, ..., T^{k-1}.

Let T be a spanning tree in G. A T-exchange is a pair of edges [e, f] such that $e \in T$, $f \notin T$, and $T-e \cup f$ is a spanning tree. The weight of T-exchange [e, f] is $w[e, f] = w(f) - w(e)$; note that the weight of tree $T-e \cup f$ is $w(T) + w[e, f]$.

Lemma 3.1[G1] A spanning tree T has minimum weight if and only if no T-exchanges have negative weight.

3.3 The Outline of the Algorithm

Our algorithm consists of routines GEN and GENK like Gabow's algorithm [G1]. GEN computes T^k when T^1, T^2, ..., T^{k-1} are given, using the branch-and-bound type technique described by Lawler [L1]. Our GEN is defferent from the Gabow's, however, in that a slightly different scheme is used to partition the solution space, and in that more information is stored in conjunction with solution space partition. GENK generates all the K minimum spanning trees using GEN as a subroutine.

The following lemma is a basis for our algorithm.

Lemma 3.2 [G1] Let T be a minimum spanning tree satisfying the constraints $IN \subset T$ and $OUT \subset E - T$, where IN and OUT are given subsets of E. Then a minimum spanning tree which is different from T and satisfies the same constraint is given by $T - e \cup f$, where [e, f] is a minimum T-exchange satisfying $e \in T - IN$ and $f \in E - T - OUT$.

Now assume that the first $k-1$ ($k > 1$) minimum spanning trees have been generated. The set of remaining spanning trees is partitioned into $k-1$ disjoint sets:

$$P_{k-1}^h = \{T^\ell \mid \ell > k-1,\ IN^h \subset T^\ell,\ OUT^h \subset E - T^\ell\},\ h = 1, 2, \ldots, k-1,$$

where IN^h and OUT^h ($1 \leq h < k$) are set of edges which will be specified later (in a way slightly different from Gabow's definition). Let for

$h = 1, 2, \ldots, k-1$,

$$Q_{k-1}^{h} = \{([e, f], r) \mid \text{For each } f \in E - T^{h} - OUT^{h},\ e \in T^{h} - IN^{h} \text{ gives the minimum } T^{h}\text{-exchange } [e, f] \text{ with weight } r = w[e, f]\}.$$

Note that each Q_{k-1}^{h} contains $|E - T^{h} - OUT^{h}| = O(m)$ labels therein.

Sets IN^{h} and OUT^{h} defining P_{k-1}^{h} $(1 \le h \le k-1)$ are given as follows. Initially when $k = 2$ (i.e., only T^{1} is obtained), IN^{1} and OUT^{1} defining P_{k-1}^{1} and Q_{k-1}^{1} are given by

$$IN^{1} = \phi \quad \text{and} \quad OUT^{1} = \phi.$$

In general, assume that T^{k} is obtained from T^{h*} by applying T^{h*}-exchange $[e^*, f^*]$. Then IN^{h} and OUT^{h} are updated as follows,

$$IN^{h*} \leftarrow IN^{h*},\ OUT^{h*} \leftarrow OUT^{h*} \cup \{f^*\},$$
$$IN^{k} \leftarrow IN^{h*} \cup \{f^*\},\ OUT^{k} \leftarrow OUT^{h*}.$$

Other IN^{h} and OUT^{h} do not change. These new sets define P_{k}^{h} for $h = 1, 2, \ldots, k$ and GEN computes the corresponding Q_{k}^{h}. Recall here that Gabow [G1] uses the scheme $IN^{h*} \leftarrow IN^{h*} \cup \{e^*\}$, $OUT^{h*} \leftarrow OUT^{h*}$; $IN^{k} \leftarrow IN^{h*}$, $OUT^{k} \leftarrow OUT^{h*} \cup \{e^*\}$. Our definition is essential to make the subsequent computation possible.

By this definition, the next lemma is obvious.

Lemma 3.3 Let k be $2 \le k \le K$.

(1) For any $h = 1, 2, \ldots, k-1$, T^h is a minimum spanning tree satisfying $IN^h \subset T^h$ and $OUT^h \subset E - T^h$, and no other T^ℓ ($\ell = 1, 2, \ldots, h-1, h+1, \ldots, k-1$) satisfy this constraint.

(2) Any spanning tree T satisfies $IN^h \subset T$ and $OUT^h \subset E - T$ for exactly one h with $1 \leq h \leq k-1$. □

When $T^1, T^2, \ldots, T^{k-1}$ are known, Lemma 3.3 (2) implies that T^k is given as a minimum spanning tree in $\cup_{h=1}^{k-1} P_{k-1}^h$ (note that P_{k-1}^h excludes $T^1, T^2, \ldots, T^{k-1}$). By Lemma 3.2 and Lemma 3.3 (2), a minimum spanning tree in P_{k-1}^h is given by $T^h - e' \cup f'$, where $([e', f'], r')$ is a label in Q_{k-1}^h with the smallest r. Thus letting $([e^*, f^*], r^*)$ be a label in $\cup_{h=1}^{k-1} Q_{k-1}^h$ with the smallest $w(T^h) + r$, T^k is given by

$$T^k = T^{h^*} - e^* \cup f^*.$$

Using these h^*, e^* and f^*, new sets P_k^h, and Q_k^h are defined and the computation proceeds as explained above.

Now we describe how to compute Q_k^h ($1 \leq h \leq k$) in GEN. While computing T^1, Q_1^1 is obtained in $O(\min(n^2, m\alpha(m, n)))$ steps by special subroutine COMPQ1 or COMPQ2 depending upon whether $n^2 < m\alpha(m, n)$ or not. These subroutines are explained in Section 3.6. When $T^k = T^{h^*} - e^* \cup f^*$ is obtained in GEN, $Q_k^{h^*}$ is computed by

$$Q_k^{h^*} = Q_{k-1}^{h^*} - \{([e^*, f^*], r^*)\}.$$

Q_k^h ($h \neq h^*$, k) are simply obtained by $Q_k^h = Q_{k-1}^h$. Finally Q_k^k is obtained by calling Subroutine COMPQ3 explained in Section 3.7. Obviously $|Q_k^h| = O(m)$ for all h. The key point is that $Q_k^{h^*}$ and Q_k^k are both obtained in O(m) steps from $Q_{k-1}^{h^*}$. Based on these Q_k^h, minimum trees in P_k^h can also be obtained in O(m) steps (note that only $P_k^{h^*}$ and P_k^k are considered since minimum spanning trees in P_{k-1}^h and P_k^h do not change for $h \neq h^*$, k). GEN is repeated for k = 2, 3, ..., K, and the entire procedure is organized as GENK.

Finally, we briefly explain the actual data structures of some of the above mentioned lists. Each set P_{k-1}^h is represented in our algorithm by a tuple

$$P_{k-1}^h = (t', [e', f'], A^h, IN^h, OUT^h, h),$$

where ([e', f'], r') is a label in Q_{k-1}^h with the smallest r, and $t' = w(T^h) + r' (= w(T^h - e' \cup f'))$. A^h is the adjacency list of T^h : $A^h = \{A^h(u) \mid u \in V\}$ and $A^h(u) = \{v \mid \text{edge } (u, v) \in T^h\}$. The length of P_{k-1}^h is O(m) since $|A^h| = O(n)$ and $|IN^h| + |OUT^h| = O(m)$. In our implementation, more information is associated with $A^h(u)$; actually it consists of the folloing tuples.

$$A^h(u) = \{(v, w(u, v), INFLAG) \mid (u, v) \in T^h\},$$

where INFLAG = 0 implies $(u, v) \notin IN^h$ and INFLAG = 1 implies $(u, v) \in IN^h$. We also prepare an adjacency list of G : $A_G = \{A_G(u) \; u \in V\}$, $A_G(u) = \{(v, w(u, v)) \mid (u, v) \in E\}$.

3.4 Algorithm for Finding K Minimum Spanning Trees

This section describes algorithms GENK and GEN in an ALGOL-like language.

Procedure GENK (G = (V, E), K); begin

comment $K \geq 2$;

1 Find the adjacency list A^1 for a minimum weight spanning tree T^1 and its weight t_1; output (A^1) ;

2 if $n^2 < m\alpha(m, n)$ then call COMPQ1 to obtain Q_1^1

else call COMPQ2 to obtain Q_1^1;

3 Find a minimum weight exchange ([e', f'], r') in Q_1^1 ;

4 Let P_1^1 be $(t_1 + r', [e', f'], A^1, \phi, \phi, 1)$;

5 For k = 2 until K do call GEN(P_{k-1}^h, Q_{k-1}^h | h = 1, 2, ..., k-1) ;

end GENK ;

Procedure GEN(P_{k-1}^h, Q_{k-1}^h | h = 1, 2, ..., k-1) ; begin

1 Find $P_{k-1}^{h*} = (t^*, [e^*, f^*], A^{h*}, IN^{h*}, OUT^{h*}, h^*)$ with the smallest weight t among $P_{k-1}^h = (t, [e', f'], A^h, IN^h, OUT^h, h)$, h = 1, 2, ..., k-1 ;

2 if $t^* = \infty$ then stop (all spanning trees have been output and G has only k-1 spanning trees) ;

else begin

3 $A^k \leftarrow A^{h*}$ with edge e^* replaced by f^*(A^k is the adjacency list of T^h) ; output (A^k) ;

4 $Q_k^{h*} \leftarrow Q_{k-1}^{h*} - \{([e^*, f^*], t^* - t_{h*})\}$;

5 Call COMPQ3(A^k, IN^k, OUT^k, f^*, Q_{k-1}^{h*}) to obtain Q_k^k ;

6 $Q_k^h \leftarrow Q_{k-1}^h$ for $h \neq h^*, k$;

7 if $Q_k^{h*} \neq \phi$ then $P_k^{h*} \leftarrow (t_{h*} + r', [e', f'], A^{h*}, IN^{h*}, OUT^{h*} \cup f^*, h^*)$, where $([e', f'], r')$ is a label in Q_k^{h*} with the minimum r and t_{h*} (the weight of T^{h*}) can be computed by $t_{h*} = t^* - w[e^*, f^*]$.

else $P_k^{h*} \leftarrow (\infty, \phi, \phi, \phi, \phi, h^*)$;

8 if $Q_k^k \neq \phi$ then $P_k^k \leftarrow (t^* + r'', [e'', f'',], A^k, IN^{h*} \cup f^*, OUT^{h*}, k)$, where $([e'', f''], r'')$ is a label in Q_k^k with the minimum r.

else $P_k^k \leftarrow (\infty, \phi, \phi, \phi, \phi, h^*)$;

9 $P_k^h \leftarrow P_{k-1}^h$ for $h \neq h^*, k$;

end

return

end GEN ;

3.5 The Correctness and the Time Bound of the Algorithm

Lemma 3.4 For each $k = 2, 3, \ldots, K$, GEN correctly computes T^k, Q_k^h and P_k^h $(h = 1, 2, \ldots, k)$ in $O(m)$ steps.

Proof Since the correctness follows from the result of [G1] and the discusstion given so far, we consider the time requirement only. Line 1 of GEN finds P_{k-1}^{h*} with the minimum t among P_{k-1}^h, $h = 1, 2, \ldots, k-1$. This is done in $O(\log(k-1)) \leq O(\log K) = O(m)$ steps (since the number of spanning trees $\leq 2^m$) if an appropriate sorting technique is used (e.g., heap sort [K16]) for the set $\{P_{k-1}^h \mid h = 1, 2, \ldots, k-1\}$. Line 2 requres constant steps. Line 3 is done in $O(n)$ $(\leq O(m))$ steps by $|A^{h*}| = O(n)$. Line 4 requires constant steps. Line 4 requires $O(m)$ steps since $|Q_{k-1}^{h*}| = O(m)$. Line 5 calls COMPQ3 and it requires $O(m)$ steps as will be shown in Section 3.7. Lines 6 and 9 require constant steps because these are accomplished simply by keeping the previous data. Lines 7 and 8 are done in $O(m)$ steps by $|Q_k^{h*}| = O(m)$ and $|Q_k^k| = O(m)$. (Adjustment of data structure of $\{P_k^h \mid h = 1, 2, \ldots, k\}$ (e.g., using heap) is also done in $O(\log k) \leq O(m)$ steps, as is well known.) Thus all computation in GEN is done in $O(m)$ steps. □

Theorem 3.5 GENK correctly generates the K minimum spanning trees from T^1 to T^K in $O(Km + \min(n^2, m \log\log n))$ time.

Proof The correctness of GENK follows from the previous

discussion. Time requirement is analyzed here. Line 1 requires $O(\min(n^2, m \log \log n))$ steps (e.g., [C3], [Y1]). Line 2 requires $O(\min(n^2, m\alpha(m, n))$ steps as shown in Section 3.6. Line 3 requires $O(m)$ steps by $|Q_1^1| = O(m)$. Line 4 requires constant time. Line 5 calls GEN K-1 times, and requires $O(Km)$ steps in total by Lemma 3.4. Thus the total time is as shown above. □

A straightforward implementation of GENK requires $O(Km)$ space mainly to store Q_{k-1}^h and P_{k-1}^h for $h = 1, 2, \ldots, k-1$. This will be reduced to $O(K+m)$ in Section 3.8.

3.6 Subroutines COMPQ1 and COMPQ2

This section briefly explains two subroutines COMPQ1 and COMPQ2, computing Q_1^1 in $O(n^2)$ steps and in $O(m\,\alpha(m, n))$ steps respectively. These are based on the next lemma.

Lemma 3.6 For a given edge $f \in E - T^1$, let e be the maximum weight edge ($\neq f$) on the unique cycle formed by adding f to T^1. Then [e, f] is the smallest T^1 - exchange with the given $f \in E - T^1$.

Proof Immediately follows since the weight of edge exchange w[e, f] is given by w(f) - w(e). □

Q_1^1 is therefore computed by finding [e, f] and r = w[e, f] of Lemma 3.6 for every edge $f = (u, v) \in E - T^1$. COMPQ1 is first outlined. It is a slight augmentation of Prim's algorithm [P4] which computes T^1 in $O(n^2)$ time; we assume reader's familiarity with Prim's algorithm. Consider a computation stage when an edge (x, v) is added to the current fragment (subtree which is going to comprise T^1), where x is a vertex in the fragment and v not in the fragment. For each vertex u in the fragment, let (h, k) be the maximum weight edge in $u \xrightarrow[T^1]{*} x$ (which has already been computed and stored). Then a maximum weight edge in $u \xrightarrow[T^1]{*} v$ for $f = (u, v)$ is obtained by taking the edge with the larger weight between (x, v) and (h, k). (This property easily follows from Lemma 3.6 and is omitted.) Thus computation of such maximum weight edges

for all (u, v), such that u's are vertices in the fragment, is done in $O(n)$ time. Since this can be repeated n-1 times until T^1 is constructed by Prim's algorithm, computing the maximum weight edge for every $f = (u, v) \in E$, the total time to compute Q_1^1 is $O(n^2)$.

Theorem 3.7 COMPQ1 computes Q_1^1 in $O(n^2)$ time. □

COMPQ2 is a straightforward adaptation of Tarjan's algorithm [T3] for verifying in $O(m\,\alpha(m, n))$ time that a tree T in an undirected graph is a minimum spanning tree. His algorithm involves the computation of the maximum weight edge along the path $u \xrightarrow[T]{*} v$, for each edge $f = (u, v) \notin T$. By Lemma 3.6, this portion of his algorithm can be directly used as COMPQ2 for computing Q_1^1.

Theorem 3.8 COMPQ2 computes Q_1^1 in $O(m\,\alpha(m, n))$ time. □

3.7 Subroutine COMPQ3

This section describes Subroutine CPOMPQ3(A^k, IN^k, OUT^k, f*, Q_{k-1}^{h*} for obtaining Q_k^k in O(m) steps when $T^k = T^{h*} - e^* \cup f^*$ is given, where [e*, f*] is the minimum T^{h*} - exchange in $\bigcup_{h=1}^{k-1} Q_{k-1}^h$ (see Section 3.3). COMPQ3 is based on the following lemma.

Lemma 3.9 For T^k and the edge $f^* = (u^*, v^*) \in T^k$ defined above, let $T^k(u^*)$ and $T^k(v^*)$ be two trees obtained from T^k by deleting f*, where $u^* \in V^k(u^*)$ and $v^* \in V^k(v^*)$. Here $V^k(x)$ is the set of vertices in the connected component $T^k(x)$. Let f = (u, v) be an edge in $E - T^k - OUT^k$.

(1) If u, $v \in V^k(u^*)$ or u, $v \in V^k(v^*)$, the label ([e, f], r) stored in Q_{k-1}^{h*} is also in Q_k^k.

(2) If $u \in V^k(u^*)$ and $v \in V^k(v^*)$, then ([e, f], r) stored in Q_k^k is determined by

$$w(e) = \max[\max\{w(g) \mid g \notin IN^k,\ g \text{ is on } u^* \xrightarrow[T^k(u^*)]{*} u\},$$

$$\max\{w(h) \mid h \notin IN^k,\ h \text{ is on } v^* \xrightarrow[T^k(v^*)]{*} v\}],$$

$$r = w(f) - w(e).$$

Proof (1) Assume u, $v \in T^k(u^*)$ without loss of generality (see Fig. 3.1). Since $T^k - f^* = T^{h*} - e^*$ (i.e., $T^k(u^*) = T^{h*}(u^*)$), it holds that $u \xrightarrow[T^k]{*} v = u \xrightarrow[T^{h*}]{*} v$, and f* is not an edge on

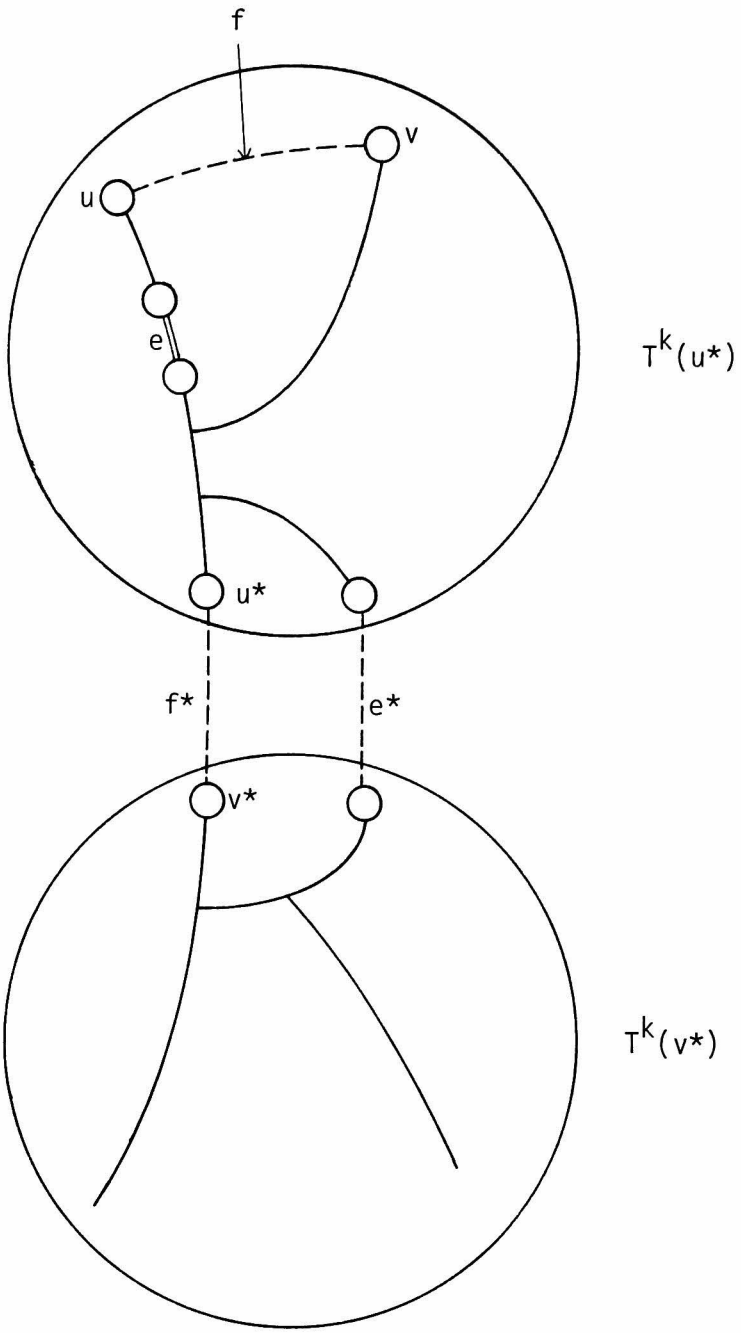

Fig. 3.1 Illustration of T^k-exchange in the proof of Case (1) of Lemma 3.9.

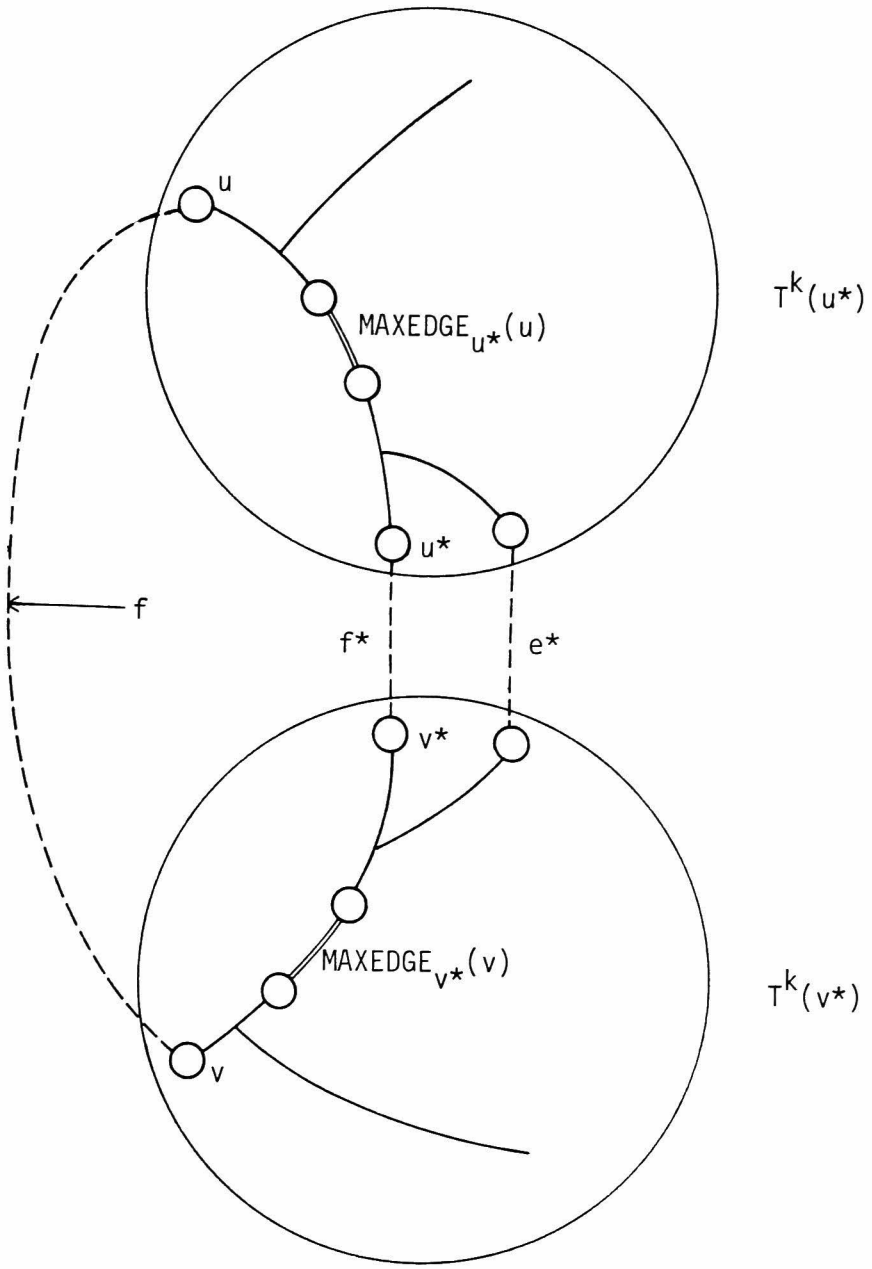

Fig. 3.2 Illustration of T^k-exchange in the proof of Case (2) of Lemma 3.9.

$u \xrightarrow[T^k]{*} v$. Thus by Lemma 3.6 the label ([e, f], r) stored in Q_{k-1}^{h*} is also in Q_k^k.

(2) Since $u \xrightarrow[T^k]{*} v$ is equal to $u \xrightarrow[T^k(u*)]{*} u* \longrightarrow v* \xrightarrow[T^k(v*)]{*} v$ and $(u*, v*)(=f*) \in IN^k$ (see Fig. 3.2), the edge e defining $([e, f], r) \in Q_k^k$ is the maximum weight edge $\notin IN^k$ on either $u \xrightarrow[T^k(u*)]{*} u*$ or $v* \xrightarrow[T^k(v*)]{*} v$ by Lemma 3.6. □

To compute $([e, f], r) \in Q_k^k$ efficiently by Lemma 3.9, COMPQ3 preprocesses Trees $T^k(u*)$ and $T^k(v*)$ by calling subroutines EDGEFIND ($\tilde{A}^k$, u*, IN^k) and EDGEFIND($\tilde{A}^k$, v* IN^k), where $\tilde{A}^k$ is the adjacency list of $T^k(u*) \cup T^k(v*)$. EDGEFIND($\tilde{A}^k$, u*, IN^k) finds the maximum weight edge $MAXEDGE_{u*}(u) \in T^k(u*) - IN^k$ on $u* \xrightarrow[T^k(u*)]{*} u$ for each vertex $u \in T^k(u*)$. Its weight is stored in $W_{u*}(u)$. (Subscript u* is added to indicate MAXEDGE and W obtained by EDGEFIND($\tilde{A}^k$, u*, IN^k).) EDGEFIND($\tilde{A}^k$, v*, IN^k) is similar. After this, ([$MAXEDGE_{w*}(z)$, f], $w(f) - W_{w*}(z)$) is added to Q_k^k for each edge $f = (u, v) \in E - T^k - OUT^k$ with $u \in V^k(u*)$ and $v \in V^k(v*)$, where $W_{w*}(z) = \max\{W_{u*}(u), W_{v*}(v)\}$.

For each $f = (u, v) \in E - T^k - OUT^k$ with $u, v \in V^k(u*)$ or $u, v \in V^k(v*)$, ([e, f], r) in Q_{k-1}^{h*} is directly stored in Q_k^k.

Procedure COMPQ3(A^k, IN^k, OUT^k, $f* = (u*, v*)$, Q_{k-1}^{h*}); begin

1 $Q_k^k \leftarrow \phi$; $\tilde{A}^k(u*) \leftarrow A^k(u*) - \{v*\}$; $\tilde{A}^k(v*) \leftarrow A^k(v*) - \{u*\}$; $\tilde{A}^k(u) \leftarrow A^k(u)$ for $u \neq u*, v*$; ($\tilde{A}^k = \{\tilde{A}^k(u) \mid u \in V\}$ is the adjacency list of $T^k(u*) \cup T^k(v*)$) ;

2 for $u \in V$ do

```
3       if  u∈ V^k(u*) then  N(u) ← 1 else  N(u) ← 0(N(u) is a flag
              showing whether u∈ V^k(u*) or u∈ V^k(v*)) ;
4   for u∈ V do  W_u*(u) ← W_v*(u) ← -∞, MAXEDGE_u*(u) ← MAXEDGE_v*(v) ← φ;
5   Call EDGEFIND(Ã^k, u*, IN^h) to obtain MAXEDGE_u*(u) and W_u*(u)
              for u∈ V^k(u*) ;
    Call EDGEFIND(Ã^k, v*, IN^h) to obtain MAXEDGE_v*(v) and W_v*(v)
              for v∈ V^k(v*) ;
7   for  f = (u, v) ∈ E - T^k - OUT^k do
8       if  N(u) = N(v) then  add label ([e, f], r) in Q_{k-1}^{h*} to Q_k^k
9       else add label([MAXEDGE_w*(z), f], w(f) - W_w*(z)) to Q_k^k,
              where z∈ {u, v} satisfies W_w*(z) = max(W_u*(u), W_v*(v))
              (if W_w*(z) = ∞, do not add the label since MAXEDGE_w*(z)
              does not exist) ;
    return
    end  COMPQ3 ;
    Procedure  EDGEFIND(Ã^k, p*, IN) ;begin
    Call DFS(Ã^k, φ, p*, IN) ;
    return
    end EDGEFIND

    Procedure DFS(Ã^k, x, y, IN) ;begin
1   for  z∈ Ã^k(y) - x( = Ã^k(y) if y = φ) and (y, z) ∉ IN  do
2       if  W(y) < w(y, z)  then  begin
                                    MAXEDGE(z) ← (y, z) ; W(z) ← w(y, z) ;
                                  end
```

```
3       else begin
                MAXEDGE(z) ← MAXEDGE(y) ; W(z) ← W(y) ;
            end
4       Call DFS(Ã^k, y, z, IN) ;
    return
    end DFS ;
```

Theorem 3.10 COMPQ3(A^k, IN^k, OUT^k, f*, Q_{k-1}^{h*}) correctly computes Q_k^k in O(m) steps.

Proof It is obvious that $\tilde{A}^k = \{\tilde{A}^k(u) \mid u \in V\}$ obtained from A^k at line 1 is the adjacency list of $T^k(u^*) \cup T^k(v^*)$. Thus EDGEFIND($\tilde{A}^k$, u*, IN^k) and EDGEFIND($\tilde{A}^k$, v*, IN^k) correctly compute $MAXEDGE_{u^*}(u)$, $W_{u^*}(u)$ for all $u \in V^k(u^*)$ and $MAXEDGE_{v^*}(v)$, $W_{v^*}(v)$ for all $v \in V^k(v^*)$. Thus lines 7-9 correctly compute Q_k^k by Lemma 3.9. Next we analyze time requirement. Line 1 requires O(n) steps since $|A^k| = O(n)$. Lines 2 and 3 are done in O(n) steps by computing $V^k(u^*)$ and $V^k(v^*)$ using A^k, and associating flag N(u) to $u \in V$ by using $\tilde{A}^k$. Line 4 is also done in O(n) steps. Lines 5 and 6 require O(n) steps, by $|V^k(u^*)| = O(n)$, $|V^k(v^*)| = O(n)$ and $|A^k| = O(n)$. To execute lines 7-9 in constant time for each $f = (u, v) \in E - T^k - OUT^k$, note that

$$E - T^k - OUT^k = \{f \mid ([e, f], r) \in Q_{k-1}^{h*}\} - f^* \cup e^*$$

holds. Furthermore $N(u) \neq N(v)$ holds for $e^* = (u, v)$. Thus adding label ([e, f], r) in Q_{k-1}^{h*} to Q_k^k at line 8 is done in constant time

if $f(\neq e^*)$ at line 7 is directly taken from $Q_{k-1}^{h^*}$ (i.e., constant time is required to search ([e, f], r) in $Q_{k-1}^{h^*}$ at line 8). Line 8 obviously requires constant time. Thus lines 7-9 require O(m) steps in total by $|E| = m$. □

3.8 Space Reduction

The required space for GENK is reduced to $O(K+m)$ in this section, although GENK explained in Section 3.4 requires $O(Km)$ space to store P^h_{k-1} and Q^h_{k-1} $(h = 1, 2, \ldots, k-1)$ (space required for other data is obviously $O(m)$).

First, in order to reduce space requirement of P^h_{k-1} from $O(Km)$ to $O(K+m)$ we modify the data structure representing P^h_{k-1} in almost the same way as done by Gabow [G1]. Namely, the data structure of P^1_{k-1} is the same as the one discussed in Section 3.3 (P^1_{k-1} requires $O(m)$ space). P^h_{k-1} $(h > 1)$ are modified to

$$P^h_{k-1} = (t', [e', f'], [e^*(h), f^*(h)], h^*(h), b(h), s(h), h),$$
$$h = 2, 3, \ldots, k-1,$$

(thus P^h_{k-1} $(h = 2, 3, \ldots, k-1)$ require $O(k) \leq O(K)$ space), where t', $[e', f']$ are defined in Section 3.3, and T^h is obtained from $T^{h^*(h)}$ by $T^{h^*(h)}$ - exchange $[e^*(h), f^*(h)]$. The derivation of $T^2, T^3, \ldots,$ from T^1 is represented by a rooted tree ; T^{h^*} is a father of T^h (or T^h is a son of T^{h^*}) if T^h is derived from T^{h^*} by a T^{h^*} - exchange, and T^h and T^ℓ are brothers if their fathers coincide. T^h is placed to the left of its brother T^ℓ if $h < \ell$. $b(h)$ and $s(h)$ in P^h_{k-1} denote the brother $T^{b(h)}$ immediately to the left of T^h ($b(h) = 0$ if T^h is the leftmost brother) and the rightmost son $T^{s(h)}$. Obviously, T^1 is the root of this tree. Based on the new lists, T^h, IN^h and OUT^h can be constructed in

$O(m)$ time by following the path from T^h up to root T^1. This technique is almost the same as Gabow's, and hence the details are omitted. The rest of computation is then applied to the reconstructed P^h_{k-1}.

In order to reduce the space requirement of Q^h_{k-1} ($h = 1, 2, \ldots, k-1$) to $O(K)$ we execute the following clean-up step from time to time. Note that each edge T^h - exchange $([e, f], r) \in Q^h_{k-1}$ induces a spanning tree $T^h - e \cup f$ with weight $w(T^h) + r$. However only $K-(k-1)$ smallest spanning trees induced from $\cup_{h=1}^{k-1} Q^h_{k-1}$ are necessary to compute T^h, T^{h+1}, ..., T^K, as justified below. Therefore the clean-up step removes all $([e, f], r)$'s from $\cup_{h=1}^{k-1} Q^h_{k-1}$ except those with $K-(k-1)$ smallest $w(T^h) + r$'s. Clean-up step is done in $O(K')$ steps, where $K' = |\cup_{h=1}^{k-1} Q^h_{k-1}|$, by finding the $K-(k-1)$th smallest element in $O(K')$ steps by the fast algorithm [B6], and then removing $([e, f], r)$'s with larger $(w(T^h) + r)$'s.

The clean-up step is justified as follows. Suppose that a T^h - exchange $[e, f]$ corresponding to $f \in E - T^h - OUT^h$ is removed from $Q^h_{h'-1}$ by a clean-up step. Later a T^k may be obtained from T^h by $T^h - e^* \cup f^*$, and Q^k_k is computed from Q^h_{k-1} by COMPQ3. Note that Q^h_{k-1} is obtained from $Q^h_{h'-1}$ and therefore Q^k_k does not contain the minimum T^k - exchange $[e', f]$ corresponding to the removed f. At this point, it is necessary to show that such $[e', f]$ in Q^k_k can be ignored for the rest of computation. However, this is obvious because $w(T^h - e \cup f) \leq w(T^k - e' \cup f)$ can be easily proved and hence

$T^k - e' \cup f$ is not a member of the K minimum spanning trees.

Now execute the clean-up step whenever $K' > 2K$ is satisfied. Then $K' = O(K)$ always holds. Since K' increases at most by m at every iteration, the clean-up step is necessary at every $\lceil K/m \rceil$ itertations, where $\lceil x \rceil$ denotes the smallest integer not smaller than x. Hence the clean-up step is executed $O(K) \cdot O(m/K) = O(m)$ times before the entire computation is completed. Therefore $O(Km)$ steps are required for the clean-up computation, but the time boud of the entire algorithm does not change.

Consequently, we have the next theorem.

Theorem 3.11 GENK requires $O(Km + \min(n^2, m \log \log n))$ time and $O(K+m)$ space. □

3.9 Conclusion

This chapter has proposed an efficient algorithm for finding K minimum spanning trees in an undirected graph. This requires $O(Km + \min(n^2, m\log\log n))$ running time and $O(K + m)$ storage space. This method is superior to the existing methods, the best of which requires $O(Km\alpha(m, n) + m\log n)$ running time and $O(K + m)$ storage space. The efficiency improvement was attained by developing subroutines to find in $O(\min(m\alpha(m, n), n^2))$ time the second minimum spanning and to find in $O(m)$ time the k-th minimum spanning tree for each $k = 3,4,\cdots$.

Finally, it should be mentioned that the basic idea for the enumeration and space reduction developed in this chapter will also be used in Chapter 9 which proposes an algorithm for obtaining K best solutions of a simple resource allocation problem.

CHAPTER 4

AN EFFICIENT ALGORITHM FOR THE SIMPLE RESOURCE ALLOCATION PROBLEM

This chapter deals with the simple resource allocation problem. This problem is to minimize an objective function represented by a sum of separable convex functions under only one constraint that a sum of integer variables is equal to a given integer N. In addition to dynamic programming approach, some algorithms have been known for this problem. This chapter presents a simple and efficient algorithm based on the Lagrange multiplier method, requiring $O(n^2 \log^2 N)$ running time, where n is the number of the variables. This algorithm is faster than already existing algorithms if N is much larger than n.

4.1 Introduction

This chapter discusses the following simple resource allocation problem.

$$\text{P:}\quad \text{Minimize} \sum_{i=1}^{n} f_i(x_i)$$

$$\text{subject to} \sum_{i=1}^{n} x_i = N \qquad x_i\text{: nonnegative integers,}$$

where each f_i is a convex function defined over [0, N] and N is a given positive integer. This problem arises in various appli-

cation areas whenever it is asked to allocate a fixed amount of the resources in an optimal way.

Properties and algorithms of this problem have been discussed in numerous papers over the last years [D3, D4, E1, F1, F4, G2, G5, H2, J2, K1, K3, K17, M1, M4, M5, S1, S5, S7, V1, V2, W1, W3]. Dynamic programming (DP) is one of the methods to solve this problem, which is applicable even if f_i's are not convex. Applying DP to this problem, its multi-stage decision process can be represented by a directed graph. Then, the problem can be regarded as the shortest path problem discussed in Chapter 2. Since the corresponding directed graph has O(nN) vertices, an optimal solution is obtained in $O(nN^2)$ running time as proved in [I4].

Besides DP approach, however, more efficient algorithms have been known fow this problem. The incremental method is one of them [G5, S3]. It requires O(n + N log n) running time. However, its time bound is exponential(strictly speaking, pseudo polynomial [G3]) if measured against the length of input data which is O(n + log N).

This chapter proposes a new algorithm based on the Lagrange multiplier method. It requires $O(n^2 \log^2 N)$ time. This time bound is a polynomial in the length of the input data. The implication of this result is twofold. First, as an algorithm for practical use, the new algorithm should be faster than the

previous ones at least when N is much larger than n (one particular example has been discussed in [I10]). Secondly, the complexity issue concerning the resource allocation problem is resolved. That is, the resource allocation problem turns out to belong to class $\mathcal{P}$ (polynomially solvable [A1, G4, K3]), although a similar problem called the knapsack problem is NP-complete [L5].

This chapter is organized as follows. Section 4.2 describes notations and basic properties. Section 4.3 gives the algorithm for solving the resource allocation problem P in $O(n^2 \log^2 N)$ time. Section 4.4 proves the correctness and analyzes the time bound of the algorithm.

4.2 Notations and Basic Concepts

For $i = 1, 2, \ldots, n$, convex functions $f_i(x_i)$ are defined over the interval $[0, N]$. For an integer x_i with $i = 1, 2, \ldots, n$ and $0 \le x_i \le N$, let

$$d_i(x_i) = f_i(x_i + 1) - f_i(x_i). \tag{4.1}$$

Here $d_i(-1) = -\infty$ and $d_i(N) = \infty$ are assumed by convention. Note that $d_i(-1) \le d_i(0) \le \ldots \le d_i(N)$ holds by convexity of $f_i(x_i)$. Let $\hat{f}_i(x_i)$ be the convex function over $[0, N]$, which is the piecewise linear approximation of $f_i(x_i)$ on integer lattice:

$$\begin{aligned} \hat{f}_i(x_i) &= f_i(x_i) && \text{if } x_i \text{ is an integer,} \\ &= f_i(\lfloor x_i \rfloor) + (x_i - \lfloor x_i \rfloor)[f_i(\lfloor x_i \rfloor + 1) - f_i(\lfloor x_i \rfloor)] && \text{otherwise,} \end{aligned} \tag{4.2}$$

where $\lfloor x_i \rfloor$ is the largest integer not greater than x_i. The optimal solution of P does not change even if $f_i(x_i)$ is replaced by $\hat{f}_i(x_i)$. Therefore we assume the following relation throughout this chapter.

$$f_i(x_i) = \hat{f}_i(x_i), \qquad \text{for } i = 1, 2, \ldots, n.$$

Let $x^* = (x_1^*, \ldots, x_n^*)$ denote an optimal solution of P and $x' = (x_1', \ldots, x_n')$ denote an optimal solution of P' which is P with the integrality condition of x_i is dropped. The subgradient $\partial f(x)/\partial x$ of f at x is defined by

$$\frac{\partial f(x)}{\partial x} = \{\, p \mid f(z) - f(x) \geq p(z - x) \text{ for any } z \text{ in the domain of } f\}.$$

It holds by condition (4.2) (see (4.1) also) that

$$\frac{\partial f_i(x_i)}{\partial x_i} = \{\, p \mid d_i(x_i - 1) \leq p \leq d_i(x_i)\} \quad \text{if } x_i \text{ is an integer with } 0 \leq x_i \leq N$$
$$= d_i(\lfloor x_i \rfloor) \quad \text{if } x_i \text{ is not an integer.} \tag{4.3}$$

Introducing the Lagrange function

$$L(x, \lambda) = \sum_{i=1}^{n} f_i(x_i) + \lambda\Big(\sum_{i=1}^{n} x_i - N\Big),$$

we obtain the following well-known proposition (e.g., Rockafellar [R1]).

Proposition 4.1 $x' = (x_1', x_2', \ldots, x_n')$ is an optimal solution of P' if and only if (x', λ') satisfies

(1) $0 \in \dfrac{\partial L(x', \lambda')}{\partial x_i}$, $x_i' \geq 0$ for all i and

(2) $\sum_{i=1}^{n} x_i = N$. □

The first condition $0 \in \partial L(x', \lambda')/\partial x_i$ in (1) of Proposition 4.1

is equivalent to

$$\begin{aligned} -d_i(x_i') \le \lambda' \le -d_i(x_i - 1) \quad & \text{if } x_i' \text{ is an integer,} \\ \lambda' = d_i(\lfloor x_i' \rfloor) \quad & \text{if } x_i' \text{ is not an integer,} \end{aligned} \tag{4.4}$$

as easily proved from (4.3).

Theorem 4.2 There exists an optimal Lagrange mutiplier λ' of P', which is equal to $-d_i(x_i)$ for some $i \in \{1, 2, \ldots, n\}$ and $x_i \in \{0, 1, \ldots, N\}$.

Proof. Let x' be an optimal solution of P'. If x' is not an integer solution, $\lambda' = -d_i(\lfloor x_i' \rfloor)$ holds by (4.4) for some i with non-integer x_i'. Thus, assume that x' is an integer solution. Let

$$\hat{\lambda} = -d_j(x_j') = \max \{-d_i(x_i') \mid 1 \le i \le n\}.$$

It will be shown that this $\hat{\lambda}$ is optimal. First $-d_i(x_i') \le \hat{\lambda}$ for all i by definition. By condition (4.4), there exists a λ' such that $-d_i(x_i') \le \lambda'$ for all i, which implies that $\hat{\lambda} \le \lambda'$. It follws that $\hat{\lambda} \le \lambda' \le -d_i(x_i' - 1)$ for all i, and $\hat{\lambda}$ is optimal by Proposition 4.1 and (4.4). □

An optimal multiplier λ' solves not only P' but also P, as shown in the next theorem.

Theorem 4.3 For any optimal multiplier λ' of P', there

exists an integer solution $\tilde{x}$ of P' satisfying conditions (1), (2) of Proposition 4.1 (i.e., $\tilde{x}$ is an optimal solution of P).

Proof. Let $x' = (x'_1, x'_2, \ldots, x'_n)$ be an optimal solution of P' satisfying conditions (1)(2) of Proposition 4.1. Let $I = \{i \mid x'_i$ is not an integer$\}$. By condition (1) of Proposition 4.1 (i.e., (4.4) above), $\lambda' = -d_i(\lfloor x'_i \rfloor)$ hold for all $i \in I$. Then consider the following integer solution $\tilde{x}$:

$$\begin{aligned} \tilde{x}_i &= x'_i && \text{for } i \notin I \\ &= \lfloor x'_i \rfloor && \text{for } i \in I_1 \\ &= \lfloor x'_i \rfloor + 1 && \text{for } i \in I_2, \end{aligned}$$

where I_1 and I_2 are subsets of I satisfying $I_1 \cup I_2 = I$, $I_1 \cap I_2 = \phi$ and

$$\sum_{i \in I_2} (\lfloor x'_i \rfloor + 1 - x'_i) = \sum_{i \in I_1} (x'_i - \lfloor x'_i \rfloor).$$

It is clear that such I_1 and I_2 always exist, and

$$\sum_{i=1}^{n} \tilde{x}_i = N.$$

Thus $\tilde{x}$ is a feasible solution of P. Moreover condition (4.4) for x' implies

$$-d_i(\tilde{x}_i) \le \lambda' \le -d_i(\tilde{x}_i - 1) \qquad \text{for all } i.$$

This proves that $\tilde{x}$ is an optimal solution of P' by Proposition 4.1 and (4.4), and is also an optimal solution of P since $\tilde{x}$ is an integer solution. □

The algorithm given in the next section is based on these theorems. The construction of an optimal solution of P in the algorithm is however slightly different from the proof of Theorem 4.3.

4.3 Description of the Algorithm

We now describe Algorithm RAP(f_1, f_2,..., f_n, N) for solving the resource allocation problem P in $O(n^2 \log^2 N)$ steps. The basic idea to search in a systematic manner an optimal value of multiplier λ among set

$$\{-d_i(x_i) \mid i=1, 2,..., n, x_i=0, 1,..., N\}$$

as a result of Theorem 4.2. For each λ thus generated, the existence of x' satisfying conditions (1) and (2) of Proposition 4.1 is tested. If such x' exists, the current λ is output as an optimal multiplier λ^*. The test is done by computing the largest integer $x_i = \bar{x}_i$ and the smallest integer $x_i = \underline{x}_i$ satisfying $-d_i(x_i) \leq \lambda \leq -d_i(x_i - 1)$. A solution x' satisfying Proposition 4.1 exists if and only if

$$\sum_{i=1}^{n} \underline{x}_i \leq N \leq \sum_{i=1}^{n} \bar{x}_i$$

holds, as proved in Theorem 4.6. $\bar{x}_i$ and $\underline{x}_i$ are respectively computed by subroutines FMAX(i, λ) and FMIN(i, λ) in O(log N) steps by using binary search over the interval [0, N] (e.g., see [A1, K16] for the binary search technique). The search of λ is performed, as described below, in such a way that a λ satisfying

$$\sum_{i=1}^{n} \underline{x}_i \leq N \leq \sum_{i=1}^{n} \bar{x}_i$$

will be eventually obtained. The computation then halts after constructing an optimal solution x* from the obtained $\bar{x}_i$ and $\underline{x}_i$.

Initially m is set to 1 and λ is searched among the set of values $\{-d_m(0), -d_m(1),\ldots, -d_m(N)\}$ (note that $-d_m(0) \geq -d_m(1) \geq \ldots \geq -d_m(N)$ holds) by binary search which starts from the middle one $-d_m(\lfloor N/2 \rfloor)$ and changes λ as follows : λ is increased if

$$\sum_{i=1}^{n} \underline{x}_i > N,$$

and is decreased if

$$\sum_{i=1}^{n} \bar{x}_i < N.$$

If none of the λ's generated for the current m satisfies

$$\sum_{i=1}^{n} \underline{x}_i \leq N \leq \sum_{i=1}^{n} \bar{x}_i,$$

m is increased by one and the same procedure is repeated. By Theorem 4.2, λ satisfying $\Sigma \underline{x}_i \leq N \leq \Sigma \bar{x}_i$(i.e., an optimal λ) will be obtained before the search is completed for m = 1, 2,..., n.

Some notations are now introduced to give the details of RAP.

λ_k : k-th Lagrange multiplier generated during computation.

$\underline{t}$, $\bar{t}$: variables used to find the next choice of λ by the rule $\lambda = -d_m(\lfloor (\underline{t} + \bar{t})/2 \rfloor)$ according to binary search. Initially $\underline{t}$ is set to 0 and $\bar{t}$ is set to N.

Algorithm RAP(f_1, f_2,..., f_n, N)

Step 1 (Initialization) Set $k \leftarrow 1$, $m \leftarrow 1$.

Step 2 Set $\underline{t} \leftarrow 0$, $\bar{t} \leftarrow N$, $t_k \leftarrow \lfloor(\bar{t} + \underline{t})/2\rfloor$, $\lambda_k \leftarrow -d_m(t_k)$.

Step 3 (Test for optimality of λ_k)

(1) For each i with $1 \le i \le n$, call FMIN(i, λ_k) to find $\underline{x}_i$ and FMAX(i, λ_k) to find $\bar{x}_i$.

(2) If

$$\sum_{i=1}^{n} \underline{x}_i^k > N,$$

then set $\bar{t} \leftarrow t_k$. Go to step 4.

(3) Else if

$$\sum_{i=1}^{n} \bar{x}_i^k < N,$$

then set $\underline{t} \leftarrow t_k$. Go to step 4.

(4) Else set $\lambda^* \leftarrow \lambda_k$. (In this case

$$\sum_{i=1}^{n} \underline{x}_i^k \le N \le \sum_{i=1}^{n} \bar{x}_i^k$$

holds and optimal Lagrange multiplier λ^* is found.)

Go to step 5.

Step 4 (Choose the next λ_{k+1}) Set $k \leftarrow k+1$, $t_k \leftarrow \lfloor(\bar{t} + \underline{t})/2\rfloor$. If $t_k = t_{k-1}$, then set $m \leftarrow m+1$ and return to step 2. Else set $\lambda_k \leftarrow -d_m(t_k)$ and return to step 3.

Step 5 (Find x*) Set

$$j \leftarrow \max\{\ell \mid \sum_{i=1}^{\ell} \bar{x}_i^k + \sum_{i=\ell+1}^{n} \underline{x}_i^k \le N\}.$$

For each i with $1 \le i \le j$, set $x^*_i \leftarrow \underline{x}^{-k}_i$. For each i with $j+2 \le i \le n$, set $x^*_i \leftarrow \underline{x}^{k}_{-i}$. Set

$$x^*_{j+1} \leftarrow N - \left(\sum_{i=1}^{j} \underline{x}^{-k}_i + \sum_{i=j+2}^{n} \underline{x}^{k}_{-i} \right).$$

Stop.

Subroutine FMIN(i, λ_k)

This routine finds

$$\underline{x}^{k}_{-i} = \min \{ x_i \in \{0, 1, \ldots, N\} \mid -d_i(x_i) \le \lambda_k \le -d_i(x_i - 1) \}$$

by binary search over the set [0, N], where

$$\infty = -d_i(-1) \ge -d_i(0) \ge \ldots \ge -d_i(N) = -\infty$$

holds. The detail is not given because this can be done by a direct application of the standard binary search technique. The running time is O(log N).

Subroutine FMAX(i, λ_k)

This routine finds

$$\underline{x}^{-k}_{i} = \max \{ x_i \in \{0, 1, \ldots, N\} \mid -d_i(x_i) \le \lambda_k \le -d_i(x_i - 1) \}$$

by binary search in O(log N) steps.

4.4 Correctness and Time Bound of RAP

We prove the correctness of Algorithm RAP(f_1, f_2,..., f_n, N) and analyze its running time.

Lemma 4.4 Algorithm RAP(f_1, f_2,..., f_n, N) always finds multiplier $\lambda = \lambda_k$ such that the associated $\underline{x}_i^k$ and $\bar{x}_i^k$ (i = 1, 2, ..., n) satisfy

$$\sum_{i=1}^{n} \underline{x}_i^k \leq N \leq \sum_{i=1}^{n} \bar{x}_i^k \tag{4.5}$$

Proof. Assume that λ_k is now being searched among $\{-d_m(0), -d_m(1),..., -d_m(N)\}$ for some m. If

$$\sum_{i=1}^{n} \underline{x}_i^k > N$$

holds for the current λ_k in (2) of step 3, $\bar{t}$ is set to t_k, i.e., $t_{k+1} \leq t_k$ and hence $\lambda_{k+1} \geq \lambda_k$ by $-d_m(0) \geq ... \geq -d_m(N)$. This is justified because no λ with $\lambda < \lambda_k$ satisfies (4.5) by the monotonicity $-d_m(0) \geq ... \geq -d_m(N)$. Similarly, it is justified to set $\underline{t}$ to t_k in (3) of step 3 when

$$\sum_{i=1}^{n} \bar{x}_i^k < N$$

holds. Finally if $\lambda_k = \lambda_{k-1}$ holds in step 4, m is increased by one, since this implies that no $\lambda_k = -d_m(x_m)$ ($x_m \in \{0, 1,..., N\}$) for the current m satisfies (4.5). This process eventually terminates because at least one $\lambda_k = -d_m(x_m)$ is known to satisfy

(4.5) by Theorem 4.2 (note that an optimal multiplier $\lambda_k = \lambda'$ obviously satisfies (4.5) by Theorem 4.3 since $\underline{x}_i^k \le \tilde{x}_i \le \bar{x}_i^k$ and $\sum \tilde{x}_i = N$). □

Lemma 4.5 Steps 3 and 4 of Algorithm RAP(f_1, f_2,..., f_n, N) are repeated O(log N) times for each m (tested during computation) before

$$\sum_{i=1}^{n} \underline{x}_i^k \le N \le \sum_{i=1}^{n} \bar{x}_i^k$$

holds in step 3 or $t_k = t_{k-1}$ holds in step 4.

Proof. Obvious since this is a straightforward application of binary search over the set $\{-d_m(0), -d_m(1),..., -d_m(N)\}$ with $-d_m(0) \ge -d_m(1) \ge ... \ge -d_m(N)$. □

Theorem 4.6 Algorithm RAP(f_1,..., f_n, N) correctly computes an optimal solution x* of P in $O(n^2 \log^2 N)$ steps.

Proof. (Correctness) By Lemma 4.4, the algorithm correctly obtains λ_k for which

$$\sum_{i=1}^{n} \underline{x}_i^k \le N \le \sum_{i=1}^{n} \bar{x}_i^k$$

is satisfied. We now show that x* obtained in step 5 satisfies conditions (1), (2) of Proposition 4.1. Since $\underline{x}_i^k \le x_i^* \le \bar{x}_i^k$ for all i with $1 \le i \le n$, x_i^* satisfies (1) of Proposition 4.1 from the definition of $\underline{x}_i^k$ and $\bar{x}_i^k$. It is also obvious that x* satisfies (2) of Proposition 4.1 from the way x_{j+1}^* being determined in step 5. Thus x* is an optimal solution of P' and hence of P

since all elements of x* are integers.

(Computational complexity) Step 1 requires constant time. By Lemma 4.5, steps 3 and 4 are repeated O(log N) times for each tested m before

$$\lambda_k = \lambda_{k-1} \qquad \text{or} \qquad \sum_{i=1}^{n} x_i^{-k} \leq N \leq \sum_{i=1}^{n} x_i^{k}$$

holds. Since step 3 calls FMAX and FMIN n times, it requires O(n log N) steps. Step 4 also requires constant steps (recall that each evaluation of $f_i(x_i)$ is assumed to require constant time). Therefore, the loop consisting of steps 3 and 4 requires $O(n \log^2 N)$ steps for each m. Finally, since m is increased one by one from m = 1 and step 5 is eventually reached for some m with $m \leq n$ (by Lemma 4.4), the loop consisting of steps 2, 3, 4 is repeated at most n times. Therefore, the loop of steps 2, 3, 4 requires $O(n^2 \log^2 N)$ steps in total. Step 5 requires O(n) steps. Consequently RAP($f_1, \ldots, f_n$, N) has $O(n^2 \log^2 N)$ running time. □

Note that the running time can sometimes be further reduced, if special forms of the objective function are assumed. For example the computational time becomes $O(n \log^2 N)$ if each $f_i(x_i)$ is given by $a_i f(x_i)$ for some $a_i > 0$ and a convex function $f(x_i)$ (the details are omitted).

4.5 Conclusion

This chapter has proposed an efficient algorithm for solving the simple resource allocation problem, which requires $O(n^2 \log^2 N)$ running time. After publishing this result [K6], Galil and Megiddo [G2], and Frederickson and Johnson [F4] have proposed more efficient algorithms for this problem. The latter one is faster and has running time $O(n(1 + \log(N/n)))$ if $n \leq N$, $O(n)$ otherwise. However, it whould be emphasized that our algorithm is much simpler from the viewpoint of practical implementation. It should be mentioned that the basic idea developed in this chapter will be again used in Chapter 6. Since this problem has a very simple constraint, it is therefore reasonable to generalize it in various directions. Perhaps the simplest among them is to add upper and lower bounds on variables, $\ell_i \leq x_i \leq u_i$, $i = 1,2, \cdots,n$. The proposed algorithm can be directly extended without increasing the required running time. Another generalization is to allow more than one resource constraint. However, such generalization problem does not seem to have efficient algorithms any more, as proved in the next chapter.

CHAPTER 5

A GENERALIZATION OF THE RESOURCE ALLOCATION PROBLEM

In this chapter, the simple resource allocation problem discussed in the previous chapter is generalized by allowing more than one resource constraint. This chapter proves that one type of the generalized problems can be reduced to the simple resource allocation problem discussed in the previous chapter by a certain transformation and hence the resulting problem can be easily solved by the algorithm given in Chapter 4. However, most of the generalized problems seem to become much more difficult. This chapter shows the difficulty by the failure of the incremental method which is valid for the simple problem, and by the so-called NP-hardness of the generalized problem.

5.1 Introduction

This chapter deals with the generalization of the resource allocation problem discussed in the previous chapter. That is, the problem is expressed as follows:

$$\text{S:}\quad \text{minimize } Z(X) = \sum_{i=1}^{n} f_i \left(\sum_{j=1}^{m} a_{ij} x_{ij} \right)$$

$$\text{subject to } \sum_{i=1}^{n} x_{ij} = N_j,\ j = 1, 2, \cdots, m,$$

$$x_{ij}: \text{ nonnegative integers,}$$

where f_i's are convex functions, N_j's are given positive integers and X denotes $n \times m$ matrix (x_{ij}). As discussed in Mjelde [M5] and Einbu [E2], this generalization has many applications (i.e., portfolio-selection problems, marketing problems and investment problems of weapon systems), while references [M5], [E2] treat the continuous version of problem S (i.e., the integrality condition is dropped). This chapter proves that, if $a_{ij} = 1$ for all i and j, problem S can be reduced to the simple resource allocation problem discussed in Chapter 4 and hence it can be efficiently solved. However, problem S with general value a_{ij} seems to be much more difficult. First, this chapter shows that a straightforward extension of the incremental method which is valid for the simple resource allocation problem does not work for general problem S. Next, this chapter proves that problem S belongs to the class of NP-hard problems well studied in the theory of complexity. Although details of theoretical discussion are not given here (see for example references [A1], [G4]), it is widely believed that no NP-hard problem can be generally solved with a polynomial-time algorithm.

This chapter is organized as follows. Section 5.2 provides an algorithm for the problem S, based on dynamic programming, and shows that the algorithm requires exponential running time. However, more efficient algorithms do not seem to exist as proved in the subsequent sections. Section 5.3 provides a special case

in which the problem S can be reduced to the simple resource allocation problem. Section 5.4 shows an example such that the incremental method, which is used to solve the simple resource allocation problem, cannot be immediately generalized to solve S. Section 5.5 proves NP-hardness of the problem S.

5.2 An Algorithm by Dynamic Programming

This section briefly describes an algorithm for solving the problem S, based on dynamic programming. First we define some notations. For a given m-dimensional nonnegative integer vector $y = (y_1, y_2, \ldots, y_m)$, and k with $1 \le k \le n$, let $F_k(y)$ be defined by

$$F_k(y) = \min \sum_{i=1}^{k} f_i\left(\sum_{j=1}^{m} a_{ij}x_{ij}\right)$$

$$\text{subject to } \sum_{i=1}^{k} x_{ij} = y_j, \quad j = 1, 2, \ldots, m.$$

$F_0(\cdot)$ is defined to be equal to 0 by convention.

It is easy to see that obtaining $F_n(N)$ ($N \triangleq (N_1, N_2, \ldots, N_m)$) is equivalent to solving the original problem S. Now, using the well-known "principle of optimality" (see Bellman [B5]), we obtain the following recurrence equation to compute $F_k(y)$:

$$F_k(y) = \min_{0 \le x_k \le y} \{F_{k-1}(y - x_k) + g_k(x_k)\}, \qquad (5.1)$$

where $x_k = (x_{k1}, x_{k2}, \ldots, x_{km})$ and $g_k(x_k) \triangleq f_k(\sum_{j=1}^{m} a_{kj}x_{kj})$. by using the above equation, $F_n(N)$ is computed as follows.

Step 1 Compute $F_1(y)$ for all possible y with $0 \le y \le N$.

Step 2 For each k = 1, 2, ..., n-1, compute $F_k(y)$ for all possible y with $0 \le y \le N$, by using (5.1).

Step 3 Compute $F_n(N)$ by (5.1). Halt. □

Now the computational time required for the above algorithm

is analyzed. Since the number of y satisfying $0 \leq y \leq N$ is $\Pi_{j=1}^{m}(N_j+1)$, and each $F_1(y)$ is evaluated in O(m) time, Step 1 requires $O(m\Pi_{j=1}^{m}(N_j+1))$ time. Since the number of x_k satisfying $0 \leq x_k \leq y$ is $O(\Pi_{j=1}^{m}(y_j+1))$ for each y and $g_k(x_k)$ is evaluated in O(m) time for each x_k, $F_k(y)$ is computed in $O(m\Pi_{j=1}^{m}(y_j+1))$ time. Thus the set of $F_k(y)$ for $0 \leq y \leq N$ is computed in $O(m(\Pi_{j=1}^{m}(N_j+1))^2)$ since the number of possible y with $0 \leq y \leq N$ is $O(\Pi_{j=1}^{m}(N_j+1))$. Since Step 2 is repeated n-2 times, the total required time for Step 2 is $O(nm(\Pi_{j=1}^{m}(N_j+1))^2)$. Step 3 requires $O(m(\Pi_{j=1}^{m}(N_j+1))^2)$ time as similarly proved. Therefore, the algorithm requires $O(nm(\Pi_{j=1}^{m}(N_j+1))^2)$ time in total.

It should be noted that the time required for the algorithm is exponential in the size of the input data (i.e., $O(\sum_{j=1}^{m}\log N_j + n + m)$). Therefore the algorithm becomes practically infeasible as the problem size increases. The subsequent sections prove the inherent intractability of the problem.

5.3 Special Case $a_{ij} = 1$

Assume in this section that $a_{ij} = 1$ holds for all i and j in S. Let

$$x_i \triangleq \sum_{j=1}^{m} x_{ij},\ i = 1, 2, \ldots, n \text{ and } N \triangleq \sum_{j=1}^{m} N_j$$

and consider problem P mentioned in chapter 4. Let $x^* = (x_1^*, \ldots, x_n^*)$ be an obtained optimal solution of P. An optimal solution of S, $X^* = (x_{ij}^*)$ $(1 \le i \le n,\ 1 \le j \le m)$, is then constructed from x^* as follows.

Step 1 Set $i \leftarrow 1$, $j \leftarrow 1$.

Step 2 Set $x_{ij}^* \leftarrow \min(x_i^*, N_j)$, $x_i^* \leftarrow x_i^* - x_{ij}^*$ and $N_j \leftarrow N_j - x_{ij}^*$.

Step 3 If $j < m$, then set $j \leftarrow j + 1$. Return to step 2. Else if $i < n$, then set $i \leftarrow i + 1$ and $j \leftarrow 1$ and return to step 2. Else stop.

The rationale behind this is that, in case of $a_{ij} = 1$, all resources are equally effective against each activity and hence there is no distinction between the resources to which each activity is distributed. A rigorous argument proceeds as follows.

First note that $\sum_{i=1}^{n} x_i^* = N = \sum_{j=1}^{m} N_j = \sum_{j=1}^{m} \sum_{i=1}^{n} x^*_{ij}$ holds. Also $\sum_{i=1}^{n} x_{ij}^* = N_j$ $(j=1, 2, \ldots, m,)$ holds as obvious from the way step 2 being carried out. Thus X^* is feasible in S. Fruthermore $z(x^*)$ of P and $Z(X^*)$ of S coincide, as easily shown. Since $x^* = (x_i^*)$ is optimal in P with the constraint $\sum_{i=1}^{n} x_i = N$ (i.e., $\sum_{i=1}^{n} \sum_{j=1}^{m} x_{ij}^* = N$),

which is a relaxation of the constraint of S, $\sum_{i=1}^{n} x_{ij} = N_j$ ($j = 1, 2,$ ..., m), the feasibility of X* in S and $\sum_{j=1}^{n} x^*_{ij} = x^*_i$ imply that X* is potimal in S.

The above algorithm obviously requires O(nm) computation time since one execution of Step 1, 2 or 3 requires constant time and the loop of Steps 2 and 3 are repeated nm times. Thus the total time is the time to compute x* plus O(nm) time.

It should be noted that the result is also valid when $a_{ij} = k$ for all i and j.

5.4 Failure of the Incremental Method

When a_{ij} of S are arbitrarily given, the incremental method used to solve P cannot be immediately generalized to solve S. We illustrate this by an example. A natural extension of the incremental method applied to problem S will proceed as follows.

(i) Start with the initial solution $X = (x_{11}, \ldots x_{nm}) = (0, \ldots, 0)$.

(ii) Let i_0 and j_0 be an index such that

$$Z(X(i_0, j_0,)) = \min\{Z(X(i, j)) \mid 1 \le i \le n \text{ and } \sum_{k=1}^{n} x_{kj} < N_j\}$$

where X(i, j) is X with x_{ij} increased by one. Let $x_{i_0 j_0} \leftarrow x_{i_0 j_0} + 1$

(iii) Repeat (ii) until $\sum_{k=1}^{n} x_{kj} = N_j$ are all satisfied for $j = 1, 2, \ldots, m$.

Now apply this algorithm to the following problem.

$$\text{minimize } Z(X) = (4x_{11} + 3x_{12} - 10)^2 + (3x_{21} + 2x_{22} - 14)^2$$

$$\text{subject to } x_{11} + x_{21} = 5$$

$$x_{12} + x_{22} = 3, \; x_{ij}: \text{ nonnegative integers.}$$

Then a solution $X = (x_{11}, x_{21}, x_{12}, x_{22}) = (2, 3, 1, 2)$ with $Z(X) = 2$ results by following the above incremental method. However, this is not optimal; an optimal solution is $X^* = (x^*_{11}, x^*_{21}, x^*_{12}, x^*_{22}) = (1, 4, 2, 1)$ with $Z(X^*) = 0$.

Thus some modification would be necessary to make the

incremental method work correctly. However, no algorithm with such a modification is probably as efficient as the original incremental method, in view of the NP-hardness result shown in the next section.

5.5 NP-hardness of Problem S

First recall that Problem S is actually a set of infinite problem instances defined by specifying coefficients a_{ij}, N_j and function f_i. In order to apply the NP-hardness concept, we need to modify our minimization problem S to the decision problem that asks whether a given problem instance of S has a solution X with $Z(X) \leq k$ for a given constant k. Certainly this decision problem is not harder than the corresponding minimization problem because the minimum value $Z(X^*)$ of the minimization problem immediately answers whether $Z(X) \leq k$ is possible or not. For simplicity, we also denote this decision problem by S, and prove its NP-hardness.

A decision problem A is said to be NP-hard if a problem B known to be NP-hard can be transformed to A (in polynomial time) in the sense that any problem instance of B has a solution if and only if the transformed instance of A has a solution. (Thus A is not easier than B since any instance of B can be solved by solving the transformed instance of A). Note that the whole set of problem instances of B may be transformed to only a subset of problem instances of A.

There are many problems known to be NP-hard. (Of course the NP-hardness of the first problem was proved by using an argument different from the above one.) Here we use the following 0-1 knapsack problem [A1, G4] as the NP-hard problem B in the above discussion:

B(I, b): Given a set of integers $I = \{a_1, a_2, \ldots, a_m\}$ and an integer b, decide whether there is a subset of I whose sum is equal to b.

For a given instance of B, i.e., a set I and an integer b, we transform it to the following problem instance of S.

Q(I, b): Given a set of integers $I = \{a_1, a_2, \ldots, a_m\}$ and an integer b, decide whether there is a solution X satisfying

$$Z(X) = \left(\sum_{j=1}^{m} a_j x_{1j} - b\right)^2 + \left(\sum_{j=1}^{m} a_j x_{2j} - b'\right)^2 \leq 0$$

$$\text{subject to } x_{1j} + x_{2j} = 1, \quad j=1, 2, \ldots, m,$$

$$x_{1j}, x_{2j}: \text{ nonnegative integers.}$$

where $b' = \sum_{i=1}^{m} a_i - b$. Note that k in the above argument is set to 0 here.

Then it is easy to see that Q(I, b) has a solution X, which is equal to saying that $\sum_{j=1}^{m} a_j x_{1j} = b$ and $\sum_{j=1}^{m} a_j x_{2j} = b'$ (recall that $x_{2j} = 1 - x_{1j}$), if and only if B(I, b) has a solution. The transformation from B(I, b) to Q(I, b) is obviously done in polynomial time. Thus S is NP-hard.

5.6 Conclusion

This chapter has considered a generalization of the simple resource allocation problem. The generalized problem is, in general, much more difficult than the simple one, though a special class of the problem can be reduced to the simple one considered so far. The difficulty has been proved by showing the failure of incremental method and by the NP-hardness of the generalized problem. Nevertheless, it seems meaningful to develop a practically feasible algorithm, since this problem has many application fields as mentioned in Section 5.1. One of the possible approaches is the application of dynamic programming. However, the straightforward application would be practically infeasible as shown in Section 5.2. Therefore, it is necessary to take advantage of specific properties of the problem in order to obtain a good algorithm. This is left for the future research.

CHAPTER 6

EFFICIENT ALGORITHMS FOR A VARIANT OF THE RESOURCE ALLOCATION PROBLEM

This chapter deals with a variant of the simple resource allocation problem discussed in Chapter 4. This problem is obtained by interchanging the objective function with the function on the left-hand side of the resource constraint in the simple resource allocation problem discussed in Chpater 4. This chapter presents three efficient algorithms for solving the problem. All of them are essentially based on the algorithms developed for the simple resource allocation problem. It is shown that each of these three algorithms is advantageous over the others according to the problem instance.

6.1 Introduction

This chapter considers the following variant of the resource allocation problem discussed in Chapter 4.

$$\text{Q: Maximize } \sum_{i=1}^{n} x_i \tag{6.1}$$

$$\text{subject to } z(x) = \sum_{i=1}^{n} f_i(x_i) \leq R \text{ and } x_i: \text{ nonnegative integers,}$$

where f_i's are nondecreasing convex functions and satisfy

$\sum_{i=1}^{n} f_i(0) \leq R$ and $\lim_{x_i \to \infty} f_i(x_i) = \infty$. These assumptions are introduced to guarantee the existence of an optimal solution. This problem is meaningful in most practical situations where the resource allocation problem discussed in Chapter 4 plays a crucial role.

This chapter presents three efficient algorithms for solving Problem Q. The first one is based on the incremental method. The second and the third ones are based on two versions of the Lagrange multiplier method developed for the simple resource allocation problem respectively. The second algorithm modifies the one presented in Chapter 4. They require $O(N^*\log n + n)$, $O(n^2 \log^2 \bar{N})$ and $O(b(n, R) + n\log n)$ time respectively, where N^* denotes the optimal objective value, $\bar{N}$ is an upper bound of N^*, and $b(n, R)$ is the time to solve the continuous version of Problem Q(i.e., integrality condition is dropped). The first algorithm is recommended if N^* is not very large compared with n; otherwise the second and the third are prefered. The third is superior to the second if $b(n, R) < O(n^2(\log\bar{N})^2)$.

This chapter is organized as follows. Section 6.2 gives notations and basic concepts. Sections 6.3 - 6.5 present respectively the first, the second and the third algorithms and analyze their running time.

6.2. Notations and Basic Concepts

Let $x^* = (x_1^*, \ldots, x_n^*)$ denote an optimal solution of Q, and let $x' = (x_1', \ldots, x_n')$ denote an optimal solution of Q', which is the same as Q except that the integrality condition on x_i's has been dropped. Let

$$d_i(x_i) = f_i(x_i + 1) - f_i(x_i), \tag{6.2}$$

where x_i is a nennegative integer. Here $d_i(-1) = 0$ is assumed for convenience. Note that $0 = d_i(-1) \le d_i(0) \le d_i(1) \le \ldots$ holds since $f_i(x_i)$ is convex and nondecreasing. We assume throughout this chapter that each $f_i(x_i)$ can be evaluated in constant time. Also we assume that $\sum_{i=1}^{n} f_i(0) < R$ and $\lim_{x_i \to \infty} f_i(x_i) = \infty$ for all i. Therefore, Q and Q' are feasible and have bounded optimal solutions. Furthermore, we sometimes assume in the subsequent discussion that $\sum_{i=1}^{n} f_i(0) < R$; if $\sum_{i=0}^{n} f_i(0) = R$ then $x_i' = \max\{x_i \mid f_i(x_i) = f_i(0)\}$ $(i = 1, 2, \ldots, n)$ obviously give an optimal solution of Q'.

The <u>subgradient</u> of f_i at x_i is a set of real numbers defined by

$$\frac{\partial f_i(x_i)}{\partial x_i} = \{p \mid f_i(z) - f_i(x_i) \ge p(z - x_i) \text{ for any } z \text{ in the domain of } f_i\}. \tag{6.3}$$

If $f_i(x_i)$ is differentiable at x_i, the subgradient $\frac{\partial f_i(x_i)}{\partial x_i}$ is equivalent to the ordinary derivative $\frac{df_i(x_i)}{dx_i}$. The following well-known

property in the convex analysis (e.g., Rockafellar [R1]) provides a basis of our algorithms.

Proposition 6.1 A solution $x' = (x_1', \ldots, x_n')$ of Q' is optimal if any only if there exists λ' satisfying

$$0 \in \frac{\partial L(x', \lambda')}{\partial x_i} \quad \text{for all } i, \tag{6.4}$$

$$x_i' \geq 0 \quad \text{for all } i, \text{ and} \tag{6.5}$$

$$\sum_{i=1}^{n} f_i(x_i') = R, \tag{6.6}$$

where $L(x, \lambda) = \sum_{i=1}^{n} x_i + \lambda(\sum_{i=1}^{n} f_i(x_i) - R)$. (6.7)

6.3 Algorithm 1

In this section we propose an algorithm for solving Q, which is based on the incremental property used in [S5, F1, V1]: let $x^{(k)} = (x_1^{(k)}, x_2^{(k)}, \ldots, x_n^{(k)})$ be an optimal solution of the problem P discussed in the previous chapter under the constraint $\sum_{i=1}^{n} x_i = k$ with $0 \le k \le N$ and let j_k be the index satisfying

$$d_{j_k}(x_{j_k}^{(k)}) = \min_{1 \le i \le n} d_i(x_i^{(k)}). \tag{6.8}$$

Then an optimal solution $x^{(k+1)}$ is given by $(x_1^{(k)}, \ldots, x_{j_k}^{(k)} - 1, x_{j_k}^{(k)} + 1, x_{j_k+1}^{(k)}, \ldots, x_n^{(k)})$.

Lemma 6.2 Let $x^{(k)}$ be an optimal solution of P under the constraint $\sum_{i=1}^{n} x_i = k$, and let $z^{(k)} = \sum_{i=1}^{n} f_i(x_i^{(k)})$. Then $x^{(k)}$ is an optimal solution of Q under the constraint $\sum_{i=1}^{n} f_i(x_i) \le r$ for any r with $z^{(k)} \le r < z^{(k+1)}$.

Proof: It is easy to see that $x^{(k)}$ is a feasible solution of Q under the constraint $\sum_{i=1}^{n} f_i(x_i) \le r$ satisfying $z^{(k)} \le r < z^{(k+1)}$. The lemma immediately follows since any solution x with $\sum_{i=1}^{n} x_i \ge k+1$ does not satisfy $\sum_{i=1}^{n} f_i(x_i) < z^{(k+1)}$ by the minimality of $z^{(k+1)}$ and the monotonicity of f_i. □

Now we give Algorithm 1, which obtains $x^{(0)}, x^{(1)}, \ldots, x^{(k)}$ by applying the above incremental method until the condition $\sum_{i=1}^{n} f_i(x_i^{(k)}) > R$ is first satisfied; $x^{(k-1)}$ is an optimal solution of Q by Lemma 6.2.

Algorithm 1

Step 1: Let $x^{(0)} \leftarrow (\overbrace{0, 0, \ldots, 0}^{n})$, $k \leftarrow 0$, $z^{(k)} \leftarrow \sum_{i=1}^{n} f_i(0)$.

Step 2: With j_k satisfying $d_{j_k}(x_{j_k}^{(k)}) = \min_{1 \le i \le n} d_i(x_i^{(k)})$, let

$$x^{(k+1)} \leftarrow (x_1^{(k)}, \ldots, x_{j_k-1}^{(k)}, x_{j_k}^{(k)} + 1, x_{j_k+1}^{(k)}, \ldots, x_n^{(k)}).$$

Step 3: Let $z^{(k+1)} \leftarrow z^{(k)} + d_{j_k}(x_{j_k}^{(k)})$ and $k \leftarrow k+1$. If $z^{(k)} > R$, then $x^* \leftarrow x^{(k-1)}$ and stop (an optimal solution is found). Else return to Step 2.

Theorem 6.3 Algorithm 1 generates an optimal solution x^* of Q in $O(N^* \log n + n)$ steps, where N^* denotes the optimal value $N^* = \sum_{i=1}^{n} x_i^*$ of Q.

Proof: Correctness is obvious by the above discussion and Lemma 6.2. Step 1 requires $O(n)$ steps. If $k = 0$, Step 2 requires $O(n)$ steps since $x^{(1)} = (0, \ldots, 1, 0, \ldots, 0)$ (1 in position j_0) is obtained in $O(n)$ steps by computing $d_i(0)$ for all i and then finding $d_{j_0}(0) = \min_{1 \le i \le n} d_i(0)$. For $k \ge 1$, Step 2 requires $O(\log n)$ steps since the index j_k of Step 2 is computed in $O(\log n)$ steps if data $d_i(x_i^{(k)})$ ($i = 1, 2, \ldots, n$) are appropriately manipulated (e.g., using the technique of 2-3 tree [A1]) by using the fact that $d_i(x_i^{(k)}) = d_i(x_i^{(k-1)})$ holds for all $i \neq j_{k-1}$ and the evaluation of each $d_i(x_i^{(k)})$ requires constant time by assumption. Since Steps 2 and 3 are repeated $N^* + 1$ times, Algorithm 1 requires $O(N^* \log n + n)$ steps in total. □

4. Algorithm 2

We assume in this section that an upper boud $\bar{N}$ of the optimal objective value of Q is given in advance. $\bar{N}$ is typically given by $\sum_{i=1}^{n} \max\{x_i \mid f_i(x_i) \le R,\ x_i\text{: integer}\}$ or by the optimal value of Q'. Let $\hat{f}_i(x_i)$ be the convex function which is the piecewise linear approximation of $f_i(x_i)$ on integer lattice points:

$$\begin{aligned} \hat{f}_i(x_i) &= f_i(x_i) \text{ if } x_i \text{ is an integer,} \\ &= f_i(\lfloor x_i \rfloor) + d_i(\lfloor x_i \rfloor)(x_i - \lfloor x_i \rfloor) \text{ otherwise,} \end{aligned} \tag{6.9}$$

where $\lfloor x_i \rfloor$ denotes the integer part of x_i. Let $\hat{Q}$ and $\hat{Q}'$ be the same as Q and Q' except that $f_i(x_i)$ is replaced by $\hat{f}_i(x_i)$ for all $i = 1, 2, \ldots, n$, and let $\hat{x}^*$ and $\hat{x}'$ be optimal solutions of $\hat{Q}$ and $\hat{Q}'$, respectively. Note that $\hat{x}^* = x^*$ holds as $\hat{Q}$ and Q are identical as far as integer solutions are concerned. As shown below, $\hat{x}^*$ is computed in $O(n^2(\log \bar{N})^2)$ steps by applying the method developed in Chapter 4 with some modification.

By (6.3) and (6.9),

$$\begin{aligned} \frac{\partial \hat{f}_i(x_i)}{\partial x_i} &= \{p \mid d_i(x_i - 1) \le p \le d_i(x_i)\} \\ &\qquad \text{if } x_i \text{ is an integer with } 0 \le x_i \le \bar{N}. \\ &= d_i(\lfloor x_i \rfloor) \\ &\qquad \text{if } x_i \text{ is not an integer.} \end{aligned} \tag{6.10}$$

Letting

$$\hat{L}(x, \lambda) = \sum_{i=1}^{n} x_i + \lambda\left(\sum_{i=1}^{n} \hat{f}_i(x_i) - R\right),$$

we obtain the following result by Proposition 6.1.

Proposition 6.4 $\hat{x}' = (\hat{x}'_1, \hat{x}'_2, \ldots, \hat{x}'_n)$ is an optimal solution of $\hat{Q}'$ if and only if there is $\hat{\lambda}'$ satisfying

$$0 \in \frac{\partial L(\hat{x}', \hat{\lambda}')}{\partial x_i} \quad \text{for all } i, \tag{6.11}$$

$$\hat{x}'_i \geq 0 \text{ for all } i, \text{ and} \tag{6.12}$$

$$\sum_{i=1}^{n} \hat{f}_i(\hat{x}'_i) = R. \tag{6.13}$$

We remark here that (6.11) is equivalent to

$$\begin{aligned} &-d_i(\hat{x}'_i) \leq 1/\hat{\lambda}' \leq -d_i(\hat{x}'_i - 1), \text{ if } \hat{x}'_i \text{ is an integer,} \\ &1/\hat{\lambda}' = -d_i(\lfloor \hat{x}'_i \rfloor), \text{ if } \hat{x}'_i \text{ is not an integer,} \end{aligned} \tag{6.14}$$

which can be easily proved from (6.10). Note that $\lambda' < 0$ follows from (6.14).

Lemma 6.5 There exists an optimal Lagrange multiplier $\hat{\lambda}'$ of $\hat{Q}'$ which is equal to $-1/d_i(x_i)$ for some i and integer x_i with $1 \leq i \leq n$ and $0 \leq x_i \leq \bar{N}$. ($-1/d_i(x_i)$ is considered to be $-\infty$ if $d_i(x_i) = 0$.)

Proof: Let $\hat{x}'$ be an optimal solution of $\hat{Q}'$. If $\hat{x}'$ is not an integer solution, $1/\hat{\lambda}' = -d_i(\lfloor \hat{x}'_i \rfloor)$ holds by (6.14) for some i with non-integer $\hat{x}'_i$. Thus assume that $\hat{x}'$ is an integer solution. Let $1/\tilde{\lambda} = -d_j(\hat{x}'_j) = \max\{-d_i(\hat{x}'_i) \mid 1 \leq i \leq n\}$ (i.e., $\tilde{\lambda} = 1/-d_j(\hat{x}'_j)$ and $\hat{x}'_j$ is

an integer). It will be shown that this $\tilde{\lambda}$ is optimal. First $-d_i(\hat{x}'_i) \le 1/\tilde{\lambda}$ holds for all i by definition. Assume $1/\tilde{\lambda} > -d_i(\hat{x}'_i - 1)$ for some i. Then $-d_i(\hat{x}'_i) \le -d_i(\hat{x}'_i - 1) < -d_j(\hat{x}'_j)$ $(= 1/\tilde{\lambda})$. This shows that the Lagrange multiplier $\hat{\lambda}'$ satisfying (6.14) for all i does not exist, a contradiction to the optimality of $\hat{x}'$ by Proposition 6.4. Thus $1/\tilde{\lambda} \le -d_i(\hat{x}'_i - 1)$ for all i, and $\tilde{\lambda}$ is optimal by Proposition 6.4 and (6.14). □

Lemma 6.6 From any optimal solution $\hat{x}'$ of $\hat{Q}'$, we can obtain an optimal solution $\hat{x}^*$ of $\hat{Q}$ satisfying $\sum_{i=1}^{n} \hat{x}^*_i = \lfloor \sum_{i=1}^{n} \hat{x}'_i \rfloor$. Also $\hat{x}^*$ is optimal to Q.

Proof: Let $\hat{x}' = (\hat{x}'_1, \hat{x}'_2, \ldots, \hat{x}'_n)$ be an optimal solution of $\hat{Q}'$ which satisfies (6.11) - (6.13). Let $I = \{i \mid \hat{x}'_i$ is not an integer$\}$. Assume $I \neq \phi$, since otherwise the lemma is proved. By (6.11) (i.e., (6.14)),

$$1/\hat{\lambda} = -d_i(\lfloor \hat{x}'_i \rfloor) \tag{6.15}$$

holds for all $i \in I$. Let $I = \{i_1, i_2, \ldots, i_m\}$, and let

$$\begin{aligned} \tilde{x}_j &= \hat{x}'_j \quad \text{for} \quad j \notin I, \\ \tilde{x}_i &= \lfloor \hat{x}'_i \rfloor + 1 \quad \text{for} \quad i \in I_1, \\ \tilde{x}_i &= \lfloor \hat{x}'_i \rfloor \quad \text{for} \quad i \in I_2, \end{aligned} \tag{6.16}$$

where I_1 and I_2 are subsets of I satisfying $I_1 \cup I_2 = I$, $I_1 \cap I_2 = \phi$ and $I_2 = \{i_1, \ldots, i_k\}$. Here $k = \max\{\ell \mid \sum_{j=1}^{\ell} (\lfloor \hat{x}'_{i_j} \rfloor + 1) + \sum_{j=\ell+1}^{m} \lfloor \hat{x}'_{i_j} \rfloor$

$\leq \sum_{i\in I} \hat{x}_i'\}$. Then

$$\sum_{i\in I} \hat{f}_i(\tilde{x}_i) = \sum_{i\in I_1} (\hat{f}_i(\lfloor \hat{x}_i' \rfloor) + d_i(\lfloor \hat{x}_i' \rfloor)) + \sum_{i\in I_2} \hat{f}_i(\lfloor \hat{x}_i' \rfloor)$$

$$= \sum_{i\in I} \hat{f}_i(\lfloor \hat{x}_i' \rfloor) - (1/\hat{\lambda}') \sum_{i\in I} (\tilde{x}_i - \lfloor \hat{x}_i' \rfloor) \text{ (by (6.15))}$$

$$\leq \sum_{i\in I} \hat{f}_i(\lfloor \hat{x}_i' \rfloor) - (1/\hat{\lambda}') \sum_{i\in I} (\hat{x}_i' - \lfloor \hat{x}_i' \rfloor)$$

(by $\sum_{i\in I}(\tilde{x}_i - \lfloor \hat{x}_i' \rfloor) = |I_1| \leq \sum_{i\in I} (\hat{x}_i' - \lfloor \hat{x}_i' \rfloor)$ and

$-1/\hat{\lambda}' \geq 0$)

$$= \sum_{i\in 1} \hat{f}_i(\lfloor \hat{x}_i' \rfloor) + \sum_{i\in I} d(\lfloor \hat{x}_i' \rfloor)\ (\hat{x}_i' - \lfloor \hat{x}_i' \rfloor)$$

$$= \sum_{i\in I} \hat{f}_i(\hat{x}_i').$$

Therefore, $\tilde{x}$ is feasible in $\hat{Q}$. Fruthermore, $\lfloor \sum_{i=1}^{n} \hat{x}_i' \rfloor = \sum_{i=1}^{n} \tilde{x}_i$ holds from the definition of $\tilde{x}$. Thus $\tilde{x}$ is an optimal solution of $\hat{Q}$. Let $\hat{x}^* = \tilde{x}$. $\hat{x}^*$ in optimal to Q also, as discussed at the beginning of this section. □

We now describe Algorithm 2. The basic idea is to search an optimal multiplier λ among set $\{-1/d_i(x_i) \mid i = 1, 2, \ldots, n, x_i = 0, 1, \ldots, \bar{N}\}$ (see Lemma 6.5). For each λ thus generated the existence of $\hat{x}'$ satisfying conditions (6.11) - (6.13) is tested. If such $\hat{x}'$ exists, it provides an optimal solution of $\hat{Q}$ (and hence of Q) by Lemma 6.6. The test is done by computing the largest integer

$x_i = \bar{x}_i$ and the smallest integer $x_i = \underline{x}_i$ satisfying $-d_i(x_i) \le 1/\lambda \le -d_i(x_i - 1)$. A solution $\hat{x}'$ exists if and only if $\sum_{i=1}^{n} \hat{f}_i(\underline{x}_i) \le R \le \sum_{i=1}^{n} \hat{f}_i(\bar{x}_i)$ holds, as proved in Theorem 6.9. $\bar{x}_i$ and $\underline{x}_i$ are respectively computed by subroutines FMAX(i, λ) and FMIN(i, λ) in $O(\log \bar{N})$ steps by using the binary search technique (e.g., see Aho et al.[A1]) over the integer set $\{0, 1, 2, \ldots, \bar{N}\}$. When $\sum_{i=1}^{n} \hat{f}_i(\underline{x}_i) \le R \le \sum_{i=1}^{n} \hat{f}_i(\bar{x}_i)$ is satisfied, an optimal solution x^* of $\hat{Q}$ (i.e., Q) is constructed from the obtained $\bar{x}_i$ and $\underline{x}_i$ in a way similar to the proof of Lemma 6.6.

The search of λ is also carried out by binary search. Initially m is set to 1, and λ is searched among the set of values $\{-1/d_m(\bar{N}) -1/d_m(1), \ldots, -1/d_m(\bar{N})\}$ (note that $-1/d_m(0) \le -1/d_m(1) \le \ldots \le -1/d_m(\bar{N})$ holds); starting with the middle one $-d_m(\lfloor \bar{N}/2 \rfloor)$, λ is decreased if $\sum_{i=1}^{n} \hat{f}_i(\underline{x}_i) > R$, and is increased if $\sum_{i=1}^{n} \hat{f}_i(\bar{x}_i) < R$. If none of the λ's generated for the current m satisfies $\sum_{i=1}^{n} \hat{f}_i(\underline{x}_i) \le R \le \sum_{i=1}^{n} \hat{f}_i(\bar{x}_i)$, m is increased by one and the same procedure is repeated. By Lemma 6.5, λ satisfying $\sum_{i=1}^{n} \hat{f}_i(\underline{x}_i) \le R \le \sum_{i=1}^{n} \hat{f}_i(\bar{x}_i)$ will be obtained before the search is completed for m = 1, 2, ..., n. Some notations are now introduced.

λ_k: The k-th Lagrange multiplier generated during computation.

$\underline{t}$, $\bar{t}$: Variables used to find the next choice of λ by the rule $\lambda = -1/d_m(\lfloor \frac{\underline{t}+\bar{t}}{2} \rfloor)$ according to binary search. Initially $\underline{t}$ is set to 0 and $\bar{t}$ is set to $\bar{N}$.

Algorithm 2 (It is assumed that $\sum_{i=1}^{n} \hat{f}_i(0) < R$; if $\sum_{i=1}^{n} \hat{f}_i(0) = R$,

Q is easily solved as mentioned in Section 6.2.)

Step 1 [Initialization]: Set $k \leftarrow 1$, $m \leftarrow 1$.

Step 2: Set $\underline{t} \leftarrow 0$, $\bar{t} \leftarrow \bar{N}$, $t_k \leftarrow \lfloor \frac{\bar{t}+\underline{t}}{2} \rfloor$, $\lambda_k \leftarrow 1/-d_m(t_k)$.

Step 3 [Test for the optimality of λ_k]:

(1) For $i = 1, 2, \ldots, n$, call FMIN(i, λ_k) to find $\underline{x}_i^k$ and call FMAX(i, λ_k) to find $\bar{x}_i^k$.

(2) If $\sum_{i=1}^{n} \hat{f}_i(\underline{x}_i^k) > R$, then set $\bar{t} \leftarrow t_k$. Go to Step 4.

(3) Else if $\sum_{i=1}^{n} \hat{f}_i(\bar{x}_i^k) < R$, then set $\underline{t} \leftarrow t_k$. Go to Step 4.

(4) Else set $\hat{\lambda}' \leftarrow \lambda_k$. (In this case $\sum_{i=1}^{n} \hat{f}_i(\underline{x}_i^k) \le R \le \sum_{i=1}^{n} \hat{f}_i(\bar{x}_i^k)$ holds and an optimal Lagrange multiplier $\hat{\lambda}'$ is found.) Go to Step 5.

Step 4 [Find λ_{k+1}]: Set $k \leftarrow k+1$, $t_k \leftarrow \lfloor \frac{\underline{t}+\bar{t}}{2} \rfloor$ If $t_k = t_{k-1}$, then set $m \leftarrow m+1$ and return to Step 2. Else set $\lambda_k \leftarrow -1/d_m(t_k)$ and return to Step 3.

Step 5 [Find x^*]: Set $j \leftarrow \max\{\ell \mid \sum_{i=1}^{\ell} \hat{f}_i(\bar{x}_i^k) + \sum_{i=\ell+1}^{n} \hat{f}_i(\underline{x}_i^k) \le R\}$ (i.e., $j=n$ or $\sum_{i=1}^{j+1} \hat{f}_i(\bar{x}_i^k) + \sum_{i=j+2}^{n} \hat{f}_i(\underline{x}_i^k) > R$). For each i with $1 \le i \le j$, set $x_i^* \leftarrow \bar{x}_i^k$. For each i with $j+2 \le i \le n$, set $x_i^* \leftarrow \underline{x}_i^k$. Set $x_{j+1}^* \leftarrow \max\{x_{j+1} \mid x_{j+1}$ is an integer such that $\hat{f}_{j+1}(x_{j+1}) \le R - (\sum_{i=1}^{j} \hat{f}_i(\bar{x}_i^k) + \sum_{i=j+2}^{n} \hat{f}_i(\underline{x}_i^k))\}$.

Subroutine FMIN(i, λ_k): This routine finds

$$\underline{x}_i^k = \min\{x_i \in \{0, 1, \ldots, \bar{N}\} \mid -d_i(x_i) \le 1/\lambda_k \le -d_i(x_i - 1)\}$$

by binary search over the set $[0, \bar{N}]$, where $0 = -d_i(-1) \ge -d_i(0) \ge \ldots \ge -d_i(\bar{N})$ holds. The detail is not given here because this can be done by a direct application of the standard binary search technique.

The running time is $O(\log \bar{N})$.

Subroutine FMAX(i, λ_k): This routine finds

$$\bar{x}_i^k = \max\{x_i \in \{0, 1, \ldots, \bar{N}\} \mid -d_i(x_i) \le 1/\lambda_k \le -d_i(x_i - 1)\}$$

by binary search in $O(\log \bar{N})$ steps.

Lemma 6.7 Algorithm 2 always finds a multiplier $\lambda = \lambda_k$ such that the associated $\underline{x}_{-i}^k$ and $\bar{x}_i^k$ $(i = 1, 2, \ldots, n)$ satisfy

$$\sum_{i=1}^{n} \hat{f}_i(\underline{x}_{-i}^k) \le R \le \sum_{i=1}^{n} \hat{f}_i(\bar{x}_i^k).$$

Proof is done in a manner similar to Lemma 4.4 in Chapter 4 by using Lemmas 6.5 and 6.6.

Lemma 6.8 Steps 3 and 4 of Algorithm 2 are repeated $O(\log \bar{N})$ times for each m (tested during computation) before $\sum_{i=1}^{n} \hat{f}_i(\underline{x}_{-i}^k) \le R \le \sum_{i=1}^{n} \hat{f}_i(\bar{x}_i^k)$ holds in Step 3 or $t_k = t_{k-1}$ holds in Step 4.

Proof: Obvious since this is a straightforward application of binary search. □

Theorem 6.9 Algorithm 2 generates an optimal solution x* of Q in $O(n^2(\log \bar{N})^2)$ steps.

Proof: [Correctness] By Lemma 6.7, the algorithm correctly obtains λ_k for which $\sum_{i=1}^{n} \hat{f}_i(\underline{x}_{-i}^k) \le R \le \sum_{i=1}^{n} \hat{f}_i(\bar{x}_i^k)$ is satisfied. This $\lambda_k = \hat{\lambda}'$ satisfies (6.11) and (6.12) of Proposition 6.4 for any x_i with $\underline{x}_{-i}^k \le x_i \le \bar{x}_i^k$, and hence any solution $\hat{x}'$ satisfying $\sum_{i=1}^{n} \hat{f}_i(\hat{x}_i') = R$

(i.e., (6.13)) and $\underline{x}_i^k \le \hat{x}'_i \le \bar{x}_i^k$ $(1 \le i \le n)$ is optimal in $\hat{Q}'$. Such $\hat{x}'$ is given for example by $\hat{x}'_i = \bar{x}_i^k$, $i = 1, 2, \ldots, j$, $\hat{x}'_i = \underline{x}_i^k$, $i = j+2, \ldots n$ and $\hat{x}'_{j+1} = \max\{x_{j+1} \mid \hat{f}_{j+1}(x_{j+1}) = R - (\sum_{i=1}^{j} \hat{f}_i(\bar{x}_i^k) + \sum_{j=j+2}^{n} \hat{f}_i(\underline{x}_i^k))\}$. Therefore the solution x^* constructed in Step 5 is optimal in $\hat{Q}$ (and hence in Q) since it is the same as $\tilde{x}$ constructed from the above $\hat{x}'$ in the proof of Lemma 6.6 (note that $I = \{j\}$ holds in this case).

[Computational Complexity] Steps 1 and 2 require constant time. By Lemma 6.8, Steps 3 and 4 are repeated $O(\log \bar{N})$ times for each tested m before $t_k = t_{k-1}$ or $\sum_{i=1}^{n} \hat{f}_i(\underline{x}_i^k) \le R \le \sum_{i=1}^{n} \hat{f}_i(\bar{x}_i^k)$ holds. Since Step 3 calls FMAX and FMIN n times, it requires $O(n \log \bar{N})$ steps. Step 4 also requires constant steps (recall that each evaluation of $\hat{f}_i(x_i)$ is assumed to require constant time). Therefore, the loop consisting of Steps 3 and 4 requires $O(n(\log \bar{N})^2)$ steps for each m. Finally, since m is increased one by one from $m = 1$ and Step 5 is eventually reached for some m with $m \le n$ (by Lemma 6.7), the loop consisting of Steps 2, 3 and 4 is repeated at most n times. Therefore, this loop requires $O(n^2(\log \bar{N})^2)$ steps in total. Step 5 requires $O(n + \log \bar{N})$ steps since an index j is obtained in $O(n)$ steps, x_i^* for $i \ne j+1$ are obtained in $O(n)$ steps and x_{j+1}^* is computed in $O(\log \bar{N})$ steps by using the binary search technique again. □

6.5 Algorithm 3

Since $f_i(x_i)$ is a nondecreasing convex function, the subgradient satisfies (e.g., [R1])

$$p_a \le p_b \text{ for any } p_a \in \frac{\partial f_i(x_i^a)}{\partial x_i} \text{ and } p_b \in \frac{\partial f_i(x_i^b)}{\partial x_i} \text{ with } x_i^a < x_i^b, \quad (6.17)$$

$$d_i(\lfloor x_i' \rfloor - 1) \le p \text{ for any } p \in \frac{\partial f_i(\lfloor x_i' \rfloor)}{\partial x_i} \text{ with } x_i' \ge 1. \quad (6.18)$$

Let x' be an optimal solution of Q'. Then we have $0 \in \frac{\partial L(x', \lambda')}{\partial x_i}$ for all i by (6.5). This implies $1 + \lambda' p' = 0$ (i.e., $p' = -1/\lambda'$) holds for some $p' \in \frac{\partial f_i(x_i')}{\partial x_i}$, and hence for all i

$$d_i(x_i) \le d_i(\lfloor x_i' \rfloor - 1) \le p \text{ (by (6.18))} \le p' \text{ (by (6.17))} = -1/\lambda' \quad (6.19)$$

for any x_i with $0 \le x_i \le \lfloor x_i' \rfloor - 1$ (let $p = p'$ if x_i' is an integer).

Similarly, the following relation also holds for all i.

$$d_i(x_i) \ge -1/\lambda' \text{ for any } x_i \text{ with } x_i \ge \lfloor x_i' \rfloor + 1. \quad (6.20)$$

An integer solution $\tilde{x}$ of Q (possibly infeasible) is now constructed from x'. Let I and J be a partition of the index set $\{1,2,\ldots,n\}$ satisfying the following conditions.

$$d_i(\lfloor x_i' \rfloor) \ge -1/\lambda' \quad \text{for } i \in I, \quad (6.21)$$

$$d_j(\lfloor x_j' \rfloor) < -1/\lambda' \quad \text{for } j \in J. \quad (6.22)$$

Then $\tilde{x}$ and $\tilde{R}$ are given by

$$\tilde{x}_i = \lfloor x'_i \rfloor \text{ for } i \in I, \tag{6.23}$$

$$\tilde{x}_j = \lfloor x'_j \rfloor + 1 \text{ for } j \in J, \tag{6.24}$$

$$\tilde{R} = \sum_{i=1}^{n} f_i(\tilde{x}_i). \tag{6.25}$$

Lemma 6.10 $\tilde{x} = (\tilde{x}_1, \ldots, \tilde{x}_n)$ is an optimal solution of Q with the constraint $\sum_{i=1}^{n} f_i(x_i) \le \tilde{R}$.

Proof: Suppose that $\tilde{x}$ is not optimal. Then there exists an optimal solution x^* such that $\sum_{i=1}^{n} x_i^* > \sum_{i=1}^{n} \tilde{x}_i$.

Case 1: $x^*_{j_1} > \tilde{x}_{j_1}$ holds for some $j_1 \in J$. If $x_i^* \ge \tilde{x}_i$ for all i with $1 \le i \le n$, x^* is not feasible since $d_{j_1}(x^*_{j_1}) \ge d_{j_1}(\tilde{x}_{j_1}) \ge -1/\lambda' > d_{j_1}(\lfloor x'_{j_1} \rfloor) \ge 0$ by (6.20) and (6.22), and hence

$$z(x^*) - \tilde{R} = z(x^*) - z(\tilde{x}) = \sum_{i=1}^{n} (f_i(x_i^*) - f_i(\tilde{x}_i)) \ge f_{j_1}(x^*_{j_1}) - f_{j_1}(\tilde{x}_{j_1}) \ge d_{j_1}(x^*_{j_1}) > 0.$$

Thus consider the following two cases.

Case 1A: Some $i_1 \in I$ satisfies $x^*_{i_1} < \tilde{x}_{i_1}$. Let x'' be the solution such that $x''_{i_1} = x^*_{i_1} + 1$, $x''_{j_1} = x^*_{j_1} - 1$ and $x''_k = x^*_k$ for $k \ne i_1, j_1$. Since $d_{i_1}(x^*_{i_1}) \le d_{i_1}(\tilde{x}_{i_1} - 1) = d_i(\lfloor x'_i \rfloor - 1) \le -1/\lambda'$ by (6.19) and $d_{j_1}(x^*_{j_1} - 1) \ge d_{j_1}(\tilde{x}_{j_1}) = d_{j_1}(\lfloor x'_{j_1} \rfloor + 1) \ge -1/\lambda'$ by (6.20), we have $z(x^*) - z(x'') = d_{j_1}(x^*_{j_1} - 1) - d_{i_1}(x^*_{i_1}) \ge 0$, i.e., x'' is feasible. Thus x'' is also optimal since x^* and x'' have the same objective value. Repeating this operation, an optimal solution satisfying the lemma will be eventually obtained.

Case 1B: Some $j_2 \in J$ satisfies $x^*_{j_1} < \tilde{x}_{j_2}$. Let x'' be a solution

defined by $x''_{j_1} = x^*_{j_1} - 1$, $x''_{j_2} = x^*_{j_2} + 1$ and $x''_k = x^*_k$ for $k \neq j_1, j_2$. Since $d_{j_1}(x^*_{j_1} - 1) \geq d_{j_1}(\tilde{x}_{j_1}) = d_{j_1}(\lfloor x'_{j_1} \rfloor + 1) \geq -1/\lambda'$ by (6.20) and $d_{j_2}(x^*_{j_2}) \leq d_{j_2}(\tilde{x}_{j_2} - 1) = d_{j_2}(\lfloor x'_{j_2} \rfloor) < -1/\lambda'$ by (6.22), $z(x^*) - z(x'') = d_{j_1}(x^*_{j_1} - 1) - d_{j_2}(x^*_{j_2}) > 0$. Thus x'' is feasible. Apply the same argument as Case 1A to x''.

Case 2: $x^*_{i_1} > \tilde{x}_{i_1}$ holds for some $i_1 \in I$. First note that it is not possible to have $x^*_i \geq \tilde{x}_i$ for all i with $1 \leq i \leq n$, as shown next. If $d_{i_1}(\tilde{x}_{i_1}) > 0$, we have $z(x^*) - \tilde{R} = z(x^*) - z(\tilde{x}) = \sum_{i=1}^{n} (f_i(x^*_i) - f_i(\tilde{x}_i)) \geq d_{i_1}(\tilde{x}_{i_1}) > 0$. Thus x^* is infeasible. On the other hand, if $d_i(\tilde{x}_i) = 0$, the solution $x^{\#} = (x'_1, \ldots, \lfloor x'_{i_1} \rfloor + 1, \ldots, x'_n)$ is also feasible in Q' (since $z(x^{\#}) - z(x') = \sum_{i=1}^{n} f_i(x^{\#}_i) - \sum_{i=1}^{n} f_i(x'_i) = f_{i_1}(x^{\#}_{i_1}) - f_{i_1}(x'_{i_1}) \leq d_{i_1}(\tilde{x}_{i_1}) = 0$) and satisfies $\sum_{i=1}^{n} x^{\#}_i > \sum_{i=1}^{n} x'_i$, a contradiction to the optimality of x'. Thus consider the following two cases.

Case 2A: Some $j_1 \in J$ satisfies $x^*_{j_1} < \tilde{x}_{j_1}$. Let x'' be a solution defined by $x''_{i_1} = x^*_{i_1} - 1$, $x''_{j_1} = x^*_{j_1} + 1$ and $x''_k = x^*_k$ for $k \neq i_1, j_1$. Since $d_{i_1}(x^*_{i_1} - 1) \geq d_{i_1}(\tilde{x}_{i_1}) \geq -1/\lambda'$ and $d_{j_1}(x^*_{j_1}) \leq d_{j_1}(\tilde{x}_{j_1} - 1) = d_{j_1}(\lfloor x'_{j_1} \rfloor) < -1/\lambda'$, $z(x^*) - z(x'') = d_{i_1}(x^*_{i_1}) - d_{j_1}(x^*_{j_1}) > 0$. Thus x'' is also feasible. Apply the argument of Case 1A to x''.

Case 2B: Some $i_2 \in I$ satisfies $x^*_{i_2} < \tilde{x}_{i_2}$. Let x'' be a solution defined by $x''_{i_1} = x^*_{i_1} - 1$, $x''_{i_2} = x^*_{i_2} + 1$ and $x''_k = x^*_k$ for $k \neq i_1, i_2$. Since $d_{i_1}(x^*_{i_1} - 1) \geq d_{i_1}(\tilde{x}_{i_1}) = d_{i_1}(\lfloor x'_{i_1} \rfloor) \geq -1/\lambda'$ and $d_{i_2}(x^*_{i_2}) \leq d_{i_2}(\tilde{x}_{i_2} - 1) = d_{i_2}(\lfloor x'_{i_2} \rfloor - 1) \leq -1/\lambda'$ by (6.19) and (6.22). Thus x' is also feasible. Apply the argument of Case 1A to x''. □

Now we give the outline of Algorithm 3. It first solves the continuous problem Q' in b(n, R) steps (e.g., $b(n, R) = O(n)$ if $f_i(x_i) = a_i x_i^k$, as easily shown). If x' is an integer solution, it is an optimal solution of Q. Otherwise it computes I, J, $\tilde{x}$ and $\tilde{R}$. This obviously requires O(n) time. If $R = \tilde{R}$, $\tilde{x}$ is an optimal solution of Q by Lemma 6.10. Finally, two cases remain.

(i) $\tilde{R} < R$. Then $\tilde{x}$ is an optimal solution to problem Q with R replaced by $\tilde{R}$. Now apply Algorithm 1 after assigning $x^{(0)}$ and $z^{(0)}$ as follows.

$$x^{(0)} \leftarrow \tilde{x}, \quad z^{(0)} \leftarrow \tilde{R}. \tag{6.26}$$

(ii) $\tilde{R} > R$. Then, starting with $x^{(0)} = \tilde{x}$ and $z^{(0)} = \tilde{R}$, apply Algorithm 1 in the reverse way according to the following modified algorithm.

Modified Algorithm 1

Step 1 Let $x^{(0)} \leftarrow \tilde{x}$, $k \leftarrow 0$, $z^{(k)} \leftarrow \tilde{R}$.

Step 2 With j_k satisfying $d_{j_k}(x_{j_k}^{(k)} - 1) = \max\{d_i(x_i^{(k)} - 1) \mid 1 \le i \le n, x_i^{(k)} \ge 1\}$, let

$$x^{(k+1)} \leftarrow (x_1^{(k)}, \ldots, x_{j_k-1}^{(k)}, x_{j_k}^{(k)} - 1, x_{j_k+1}^{(k)}, \ldots, x_n^{(k)}).$$

Step 3 Let $z^{(k+1)} \leftarrow z^{(k)} - d_{j_k}(x_{j_k}^{(k)} - 1)$ and $k \leftarrow k + 1$. If $z^{(k)} \le R$, then $x^* \leftarrow x^{(k)}$ and stop (an optimal solution is found). Else return to Stop 2.

Algorithm 3

Step 1 Given a problem Q of (6.1), obtain an optimal solution x' and the associated optimal Lagrange multiplier λ' of the continuous problem Q'. Then compute

$$\tilde{x}_i = \lfloor x_i' \rfloor \qquad \text{if } d_i(\lfloor x_i' \rfloor) \geq -1/\lambda'$$
$$= \lfloor x_i' \rfloor + 1 \qquad \text{otherwise,}$$
$$\tilde{R} = \sum_{i=1}^{n} f_i(\tilde{x}_i).$$

Step 2 (1) If $\tilde{R} = R$, then $x^* \leftarrow \tilde{x}$. Halt.

(2) If $\tilde{R} < R$, apply Algorithm 1 after replacing the initial $x^{(0)}$ and $z^{(0)}$ in Step 1 by $\tilde{x}$ and $\tilde{R}$ respectively.

(3) If $\tilde{R} > R$, apply the modified Algorithm 1.

Theorem 6.11 Algorithm 3 generates an optimal solution x^* of Q in $O(b(n, R) + n \log n)$ steps, where $b(n, R)$ is the computational time required to solve Q'.

Proof: The correctness is first proved. By Lemma 6.10, $\tilde{x}$ is an optimal solution of Q with R replaced by $\tilde{R}$. If $R = \tilde{R}$, therefore, x^* obtained in Step 2 (1) is obviously optimal. If $\tilde{R} < R$, the argument of Section 6.3 ((6.8) and Lemma 6.2) can be started from $\tilde{x}$. The optimality of x^* obtained in Step 2 (2) can be similarly proved. Finally if $\tilde{R} > R$, the argument of Section 6.3 need be reversed. We omit the details, however, since it is exactly parallel to the argument of Section 6.3.

Next we analyze the computational time. Step 1 requires $b(n, R)$

time to compute x' and $O(n)$ time to obtain $\tilde{x}$ and $\tilde{R}$. Next note that $|\sum x_i' - \sum \tilde{x}_i| \leq \sum |x_i' - \tilde{x}_i| \leq n$ holds by $|x_i' - \tilde{x}_i| \leq 1$. Furthermore $\sum x_i' \geq \sum x_i^* \geq \sum \lfloor x_i' \rfloor > \sum x' - n$; the first inequality holds because Q' is a relaxation problem of Q, and the second inequality follows from the fact that $\lfloor x' \rfloor = (\lfloor x_1' \rfloor, \ldots, \lfloor x_n' \rfloor)$ is a feasible solution of Q. Then $|\sum x_i^* - \sum \tilde{x}_i| \leq 2n$. This implies that Step 2 (2) or (3) requires at most $O(n)$ iterations of the loop consisting of Steps 2 and 3 of Algorithm 1 or the modified Algorithm 1. Since each iteration is done in $O(\log n)$ time as mentioned in Section 6.3, Step 2 of Algorithm 2 requires $O(n \log n)$ time. Thus the total computational time of Algorithm 3 is $O(b(n, R) + n \log n)$. □

6.6 Conclusion

This chapter has proposed three efficient algorithms for a variant of the simple resource allocation problem discussed in Chapter 4. It has been observed that the three algorithms respectively correspond to the three algorithms developed for the simple resource allocation problem P. Therefore it is expected that, if a new algorithm for P is developed, it will be also modified to solve the proposed variant. Recently, Frederickson and Johnson [F4] have developed an efficient algorithm for the simple resource allocation problem. It should be noted that, if this algorithm is used as the starting algorithm for constructing Algorithm 2, the running time can be reduced to $O(n\log\bar{N})$ time. However, Algorithm 2 proposed in this chapter is much simpler than the improved one from the view point of the practical implementation.

CHAPTER 7

AN EFFICIENT ALGORITHM FOR AN EQUIPOLLENT RESOURCE ALLOCATION PROBLEM

This chapter deals with an equipollent resource allocation problem, which is to allocate a given amount of resources to a given set of activities so as to minimize the maximum difference of the resulting profits between each pair of activities. This chapter proposes an efficient algorithm for this problem. The idea of the algorithm is as follows. First two related simpler problems are introduced. Secondly some properties relating optimal solutions of these two problems to an optimal solution of the original problem are discussed. Finally, based on the properties, an efficient algorithm is developed. The algorithm requires $O(n^2 + n\log N)$ running time, where n is the number of activities and N is the amount of resources.

7.1 Introduction

Chapters 4 through 6 have considered some types of resource allocation problems whose objective functions or resource constraints are represented as a sum of separable convex functions. This chapter deals with another type of resource allocation problem, which is expressed as follows.

$$\text{R:}\quad \text{minimize } z(x) = g\Big(\max_{1\le i\le n} f_i(x_i),\ \min_{1\le i\le n} f_i(x_i)\Big) \tag{7.1}$$

$$\text{subject to } \sum_{i=1}^{n} x_i = N,$$

$$\ell_i \le x_i \le u_i,\ i = 1,2,\cdots,n,$$

$$x_i:\ \text{nonnegative integers},\ i = 1,2,\cdots,n,$$

where f_i's are nondecreasing functions defined over the set of integers $\{\ell_i, \ell_i + 1, \cdots, u_i\}$, $g(\xi, \eta)$ is a nondecreasing (resp. nonincreasing) function with respect to ξ(resp. η), ℓ_i and u_i are nonnegative integers satisfying

$$\sum_{i=1}^{n} \ell_i \le N \le \sum_{i=1}^{n} u_i,$$

and N is a given positive integer. It is assumed throughout this chapter that

$$\sum_{i=1}^{n} \ell_i < N < \sum_{i=1}^{n} u_i \tag{7.2}$$

holds, since otherwise the problem is trivially solved.

This problem arises whenever it is necessary to distribute a given amount of resources to a given set of activities so as to minimize the maximum difference of the resulting profits between each pair of activities. A typical example is the seat apportionment problem, which is to allocate a given number of seats to electoral districts so that the number of seats given to each district is as proportional to its population as possible. The fairness of apportionment is for example measured by

$$f_i(x_i) = x_i/p_i$$

$$g(\xi, \eta) = \xi - \eta \text{ or } \xi/\eta,$$

where p_i is the population of district i, and x_i is the number of seats given to district i. The minimization of $g(\max_i x_i/p_i, \min_i x_i/p_i)$ leads us to define a resource allocation problem as given by (7.1).

This chapter proposes an efficient algorithm for problem R, which requires $O(n^2 + n\log N)$ running time.

This chapter is organized as follows. Section 7.2 introduces two simple problems related to problem R, and gives some properties relating optimal solutions of these problems to an optimal solution of R. Sections 7.3 - 7.6 propose an efficient algorithm for solving R based on these properties.

7.2. Properties of Optimal Solutions of R

We present in this section some properties of optimal solutions of R. For this purpose, we introduce the following problems R_a and R_b, whose optimal solutions are used to construct an optimal solution of the original problem R.

$$R_a: \quad \text{maximize } z_a(x) = \min_{1 \le i \le n} f_i(x_i) \qquad (7.3)$$
$$\text{subject to the constraint of R.}$$

$$R_b: \quad \text{minimize } z_b(x) = \max_{1 \le i \le n} f_i(x_i) \qquad (7.4)$$
$$\text{subject to the constraint of R.}$$

These problems were discussed by Jacobsen [J1], Porteus and Yormark [P4], and Brown [B7] ([B7] also treats the continuous version and includes some applications). Jacobsen's algorithm may be called the incremental or greedy method and requires $O(N \log n + n)$ time, if each evaluation of $f_i(x_i)$ is done in constant time. Porteus and Yormark's algorithm is based on similar ideas but uses binary search to find the optimal value. Brown's algorithm starts with the continuous version and requires $O(n \log n + T)$ time (T is the time to solve the continuous version of R_a and R_b). Based on the recent result [F4], we shall show in the next section that these problems can be solved in $O(n \log N)$ time.

Let x^a and x^b be optimal solutions of R_a and R_b respectively. The next lemma proved in [J1] is crucial to obtain x^a and x^b.

Lemma 7.1 A feasible solution $x = (x_1, \cdots, x_n)$ (i.e., x satisfies the constraint of R) is an optimal solutiln to R_a (resp. R_b) if x satisfies the following condition (i) (resp. (ii)).

(i) $f_j(x_j-1) \leq \min\{f_i(x_i) \mid 1 \leq i \leq n,\ x_i < u_i\}$ for any j with $x_j > \ell_j$. (7.5)

(ii) $f_j(x_j+1) \geq \max\{f_i(x_i) \mid 1 \leq i \leq n,\ x_i > \ell_i\}$ for any j with $x_j < u_j$. (7.6)

Furthermore, there exists an optimal solution x to R_a (resp. R_b) which satisfies (i) (resp. (ii)). □

For optimal solutions x^a and x^b of R_a and R_b, satisfying (7.5) and (7.6) respectively, let i_a and i_b be defined by

$$f_{i_a}(x^a_{i_a}) = \min\{f_i(x^a_i) \mid 1 \leq i \leq n,\ x^a_i < u_i\}, \tag{7.7}$$

$$f_{i_b}(x^b_{i_b}) = \max\{f_i(x^b_i) \mid 1 \leq i \leq n,\ x^b_i > \ell_i\}. \tag{7.8}$$

The following lemmas are used to obtain an optimal solution x^* of R from x^a and x^b.

Lemma 7.2 If $x^a = x^b$, then $x^* = x^a = x^b$ holds.

Proof. Obvious from the assumption on $g(\xi, \eta)$ of R. □

Lemma 7.3 $f_{i_a}(x^a_{i_a}) \leq f_{i_b}(x^b_{i_b})$ holds if $x^a \neq x^b$.

Proof. Assume otherwise. Then, for any i with $x^a_i < u_i$ and $x^b_i > \ell_i$, we have

$$f_i(x^a_i) \geq f_{i_a}(x^a_{i_a}) \text{ (by (7.7))} > f_{i_b}(x^b_{i_b}) \geq f_i(x^b_i) \text{ (by (7.8))}.$$

This implies $x^a_{i_n} > x^b_{i_n}$. For any i with $x^a_i = u_i$ or $x^b_i = \ell_i$, $x^a_i \geq x^b_i$ is obvious.

Therefore $N = \sum_{i=1}^{n} x^a_i > \sum_{i=1}^{n} x^b_i = N$, a contradiction. □

Lemma 7.4 There exists an optimal solution x* of R satisfying at least one of the following conditions.

$$\text{(a)}\quad x^*_i \geq x^a_i - 1 \quad \text{for all } i, \text{ or} \tag{7.9}$$

$$\text{(b)}\quad x^*_i \leq x^b_i + 1 \quad \text{for all } i. \tag{7.10}$$

Proof. If this is not the case, there exist k and ℓ with $x^*_k < x^a_k - 1$ and $x^*_\ell > x^b_\ell + 1$. Consider a feasible solution x' defined by $x'_k = x^*_k + 1$, $x'_\ell = x_\ell - 1$, and $x'_j = x^*_j$ for all $j \neq k, \ell$. Then we have

$$\begin{aligned} f_k(x^*_k) \leq f_k(x'_k) \leq f_k(x^a_k - 1) &\leq f_{i_a}(x^a_{i_a}) \quad \text{(by (7.5))} \\ &\leq f_{i_b}(x^b_{i_b}) \quad \text{(by Lemma 7.3)} \\ &\leq f_\ell(x^b_\ell + 1) \quad \text{(by (7.6))} \\ &\leq f_\ell(x'_\ell) \leq f_\ell(x^*_\ell). \end{aligned}$$

Thus, $z(x') \leq z(x^*)$ follows. Repeating this, the lemma will be eventually proved. □

Lemma 7.4 suggests the following procedure to obtain an

optimal x^*.

1. Compute optimal solutions x^a and x^b of R_a and R_b respectively.

2. Compute a feasible solution x^L minimizing $z(x) = g(\max f_i(x_i), \min f_i(x_i))$ among the feasible solutions satisfying $x_i \geq x_i^a - 1$ for $i = 1, 2, \cdots, n$.

3. Compute a feasible solution x^U minimizing $z(x)$ among the feasible solutions satisfying $x_i \leq x_i^b + 1$ for $i = 1, 2, \cdots, n$.

4. Finally an optimal solution x^* is obtained from x^L and x^U by

$$z(x^*) = \min\{z(x^L), z(x^U)\}.$$

The following four sections show that this approach is in fact possible.

7.3. Obtaining Optimal Solutions of R_a and R_b

Procedures for obtaining optimal solutions x^a and x^b of R_a and R_b are discussed in this section. Let

$$N_a = \sum_{i=1}^{n} u_i - N, \quad N_b = N - \sum_{i=1}^{n} \ell_i, \tag{7.11}$$

$$S_a = \{f_i(x_i) \mid \ell_i \le x_i < u_i, \ i = 1, 2, \cdots, n\}, \tag{7.12}$$

$$S_b = \{f_i(x_i) \mid \ell_i < x_i \le u_i, \ i = 1, 2, \cdots, n\} \tag{7.13}$$

By assumption (7.2), $N_a, N_b > 0$ and $S_a, S_b \neq \phi$.

Lemma 7.5 With the above notations, the next properties hold for $f_{i_a}(x^a_{i_a})$ and $f_{i_b}(x^b_{i_b})$ of (7.7) and (7.8), where x^a and x^b satisfy conditions (7.5) and (7.6) respectively.

(i) $f_{i_a}(x^a_{i_a})$ is the N_a-th largest element in S_a.

(ii) $f_{i_b}(x^b_{i_b})$ is the N_b-th smallest element in S_b.

Proof. We prove (i) only. (ii) can be similarly treated. Due to the monotonicity of f_i, $f_i(\ell_i) \le f_i(\ell_i + 1) \le \cdots \le f_i(u_i)$ holds. For an optimal solution x^a satisfying (7.5), we have

$$\sum_{i=1}^{n} (u_i - x^a_i) = \sum_{i=1}^{n} u_i - N = N_a,$$

$$f_i(x^a_i) \ge f_{i_a}(x^a_{i_a}) \text{ for any } i \text{ with } x^a_i < u_i,$$

$$f_i(x^a_i - 1) \le f_{i_a}(x^a_{i_a}) \text{ for any } i \text{ with } x^a_i > \ell_i.$$

The last property is due to Lemma 7.1. These conditions together prove property (i). □

Frederickson and Johnson [F4] have shown that the N-th largest element in an N×n matrix, in which each column is already sorted, is computed in $O(n(1+\log(N/n)))$ time if $N \geq n$, $O(n)$ time otherwise. By constructing the $N_a \times n$ matrix M_a corresponding to S_a such that its (j, i)-th element is $f_i(u_i - j)$ ($f_i(u_i - j) = -\infty$ if $u_i - j < \ell_i$), their algorithm can be used to compute the N_a-th largest element in S_a. Note that each column of M_a is already sorted by the monotonicity of f_i. The similar result also holds for S_b. Thus we have the next lemma.

Lemma 7.6 $x^a_{i_a}$ and $x^b_{i_b}$ defined in (7.7) and (7.8) for optimal solutions x^a and x^b can be computed in $O(n(1+\log(N_a/n)))$ time if $N_a \geq n$, $O(n)$ time otherwise, and $O(n(1+\log(N_b/n)))$ time if $N_b \geq n$, $O(n)$ otherwise, respectively. □

From the $x^a_{i_a}$ and $x^b_{i_b}$ computed as above, x^a and x^b can be easily constructed. For this, obtain the following k_i^+ and k_i^- for $i = 1, 2, \cdots, n$.

$$k_i^+ = \begin{cases} 0, & \text{if } f_i(u_i) < f_{i_a}(x^a_{i_a}) \\ N_a, & \text{if } f_i(u_i - N_a) > f_{i_a}(x^a_{i_a}) \\ \min\{u_i - x_i \mid f_i(x_i - 1) < f_{i_a}(x^a_{i_a}) \text{ and } x_i \geq u_i - N_a\}, & \text{otherwise,} \end{cases}$$

$$k_i^- = \begin{cases} k_i^+, & \text{if } f_i(u_i - k_i^+) \neq f_{i_a}(x_{i_a}^a) \\ \max\{u_i - x_i \mid f_i(x_i + 1) > f_{i_a}(x_{i_a}^a)\}, & \text{otherwise,} \end{cases}$$

where $f_i(x_i) = -\infty$ for $x_i < u_i - N_a$ and $f_i(x_i) = \infty$ for $x_i > u_i$ are assumed. Each k_i^+ or k_i^- can be computed in $O(\log N_a)$ time by using binary search in an obvious way. By definition

$$\sum_{i=1}^{n} k_i^+ \geq N_a \geq \sum_{i=1}^{n} k_i^-$$

holds, and k_i $(i = 1, 2, \cdots, n)$ satisfying $\sum_{i=1}^{n} k_i = N_a$ is obtained by executing the following in the order of $i = 1, 2, \cdots, n$.

$$k_i = \begin{cases} k_i^+, & \text{if } \sum_{j=1}^{i-1} k_j + k_i^+ + \sum_{j=i+1}^{n} k_j^- \leq N_a \\ k_i^-, & \text{if } \sum_{j=1}^{i-1} k_j + \sum_{j=i}^{n} k_j^- = N_a \\ N_a - \sum_{j=1}^{i-1} k_j - \sum_{j=i+1}^{n} k_j^-, & \text{otherwise.} \end{cases}$$

The computation of k_i for $i = 1, 2, \cdots, n$ is done in $O(n)$ time. Finally, an optimal solution x^a is given by

$$x_i^a = u_i - k_i, \quad i = 1, 2, \cdots, n.$$

The computation of x^b is also done in a similar manner. We call these algorithms computing x^a and x^b by MAXIMIN and MINIMAX respectively.

Lemma 7.7 Algorithm MAXIMIN (resp. MINIMAX) correctly

computes x^a (resp. x^b) in $O(n \log N_a)$ (resp. $O(n \log N_b)$) time. □

7.4. Algorithm for Obtaining x^L

For the solution x^L defined at the end of Section 7.2 corresponding to constraint (7.9), let i_L be defined as follows.

$$f_{i_L}(x^L_{i_L}) = \min_{1 \le i \le n} f_i(x^L_i) \tag{7.14}$$

Lemma 7.8 There exists an x^L such that $x^L_{i_L} = x^a_{i_L}$ or $x^a_{i_L} - 1$.

Proof. Since $x^L_{i_L} \ge x^a_{i_L} - 1$ holds by constraint (7.9), it is sufficient to derive a contradiction from $x^L_{i_L} > x^a_{i_L}$. If $x^L_{i_L} > x^a_{i_L}$, there exists x^L_k such that $x^L_k < x^a_k$. By (7.5), we have

$$f_k(x^L_k) \le f_k(x^a_k - 1) \le f_{i_a}(x^a_{i_a}) \le f_{i_L}(x^a_{i_L}) \le f_{i_L}(x^L_{i_L}).$$

By the minimality of $f_{i_L}(x^L_{i_L})$ in (7.14), this implies $f_k(x^L_k) = f_{i_L}(x^L_{i_L})$. Moreover, $x^L_k = x^a_k - 1$ holds by condition (7.9). Consequently, $f_k(x^a_k - 1) = f_k(x^L_k) = \min_i f_i(x^L_i)$. By replacing $x^L_{i_L}$ with x^L_k, we see that x^L is an optimal solution satisfying the lemma statement. □

For each k with $1 \le k \le n$ and $\alpha = 0, 1$, denote by $x^L(k, \alpha)$ a feasible solution of R, which minimizes $z(x)$ subject to

$$x_k = x^a_k - \alpha, \tag{7.15}$$

$$x_i \ge x^a_i - 1 \text{ and } f_i(x_i) \ge f_k(x_k) \text{ for all } i. \tag{7.16}$$

If $x^L(k, \alpha)$ does not exist, $z(x^L(k, \alpha))$ is set to ∞. Let $x^L(k)$ be one of the $x^L(k, 0)$ and $x^L(k, 1)$ satisfying

$$z(x^L(k)) = \min\{z(x^L(k, 0)), z(x^L(k, 1))\}. \tag{7.17}$$

Then, by Lemma 7.8, x^L is equal to one of the $x^L(k)$ such that

$$z(x^L) = \min_{1 \le k \le n} z(x^L(k)). \tag{7.18}$$

The above $x^L(k, \alpha)$, can be computed as follows.

(i) First compute for each $i \neq k$ the minimum $x_i = \underline{x}_i$ satisfying (7.16). $\underline{x}_k$ is set to $x_k^a - \alpha$. These $\underline{x}_i$ for $i \neq k$ are computed in $O(n)$ time by directly evaluating $f_i(x_i^a - 1)$, $f_i(x_i^a)$, ..., $f_i(\underline{x}_i)$ in this order ($\underline{x}_i$ is the first x_i satisfying $f_i(x_i) \ge f_k(\underline{x}_k)$) for every $i \neq k$. The total time is $O(n)$ by the property

$$\sum_{i \neq k} (\underline{x}_i - (x_i^a - 1)) \le n + \alpha - 1, \tag{7.19}$$

since otherwise

$$\sum_{i=1}^{n} \underline{x}_i > \sum_{i \neq k} (x_i^a - 1) + (x_k^a - \alpha) + (n + \alpha - 1)$$

$$= N - n - \alpha + 1 + (n + \alpha - 1) = N,$$

i.e., $x^L(k, \alpha)$ does not exist and $z(x^L(k, \alpha))$ is set to ∞. (The computation of $\underline{x}_i$ for $i \neq k$ is terminated as soon as it is detected that (7.19) does not hold, to keep the computation time within $O(n)$ even if $x^L(k, \alpha)$ does not exist.)

(ii) Compute $x^L(k, \alpha)$ by solving problem R_b in which $\ell_i = \underline{x}_i$ ($i = 1, 2, \cdots, n$) and $u_k = \underline{x}_k$ (other u_i, $i \neq k$, do not change) are used. This is because the minimization of $z(x)$ of (7.1) under (7.15) (7.16) is equivalent to the minimization of $z_b(x)$ of (7.4) under the same constraints (7.15) (7.16). By Lemma 7.6, the

N_b - th smallest element $f_{i_L}(\hat{x}_{i_L})$ in S_b is obtained in $O(n)$ time since

$$N_b = N - \sum \ell_i = N - \sum \underline{x}_i \leq N - \sum (x_i^b - 1) = n.$$

The solution $x^L(k, \alpha)$ can be constructed from $f_{i_L}(\hat{x}_{i_L})$ in $O(n)$ time by the following procedure.

(a) For $i \neq i_L$, compute $\hat{x}_i$ which is the largest x_i satisfying $f_i(x_i) < f_{i_L}(\hat{x}_{i_L})$ if $f_i(\underline{x}_i) < f_{i_L}(\hat{x}_{i_L})$; otherwise $\hat{x}_i = \underline{x}_i$. $\hat{x}_i$ is computed by evaluating $f_i(\underline{x}_i)$, $f_i(\underline{x}_i + 1), \cdots$ in this order until $f_i(x_i + 1) \geq f_{i_L}(\hat{x}_{i_L})$ is first detected.

(b) If $\sum_{i=1}^{n} \hat{x}_i < N$, take an i such that $f_i(\hat{x}_i + 1) = f_{i_L}(\hat{x}_{i_L})$ and let $\hat{x}_i \leftarrow \hat{x} + 1$. Repeat this operation until $\sum_{i=1}^{n} \hat{x}_i = N$ is attained (this is possible because $f_{i_L}(\hat{x}_{i_L})$ is the N_b-th smallest element in S_b, and $\sum_{i=1}^{n} \underline{x}_i + N_b = N$). Then $x^L(k, \alpha) = \hat{x}$ holds.

It follows from $N_b \leq n$ that the computation of (a) and (b) above is done in $O(n)$ time.

After computing $x^L(k, \alpha)$ for $\alpha = 0, 1$ by the above algorithm, $x^L(k)$ and x^L is obtained by (7.17) and (7.18). The entire algorithm is denoted XL.

Lemma 7.9 Algorithm XL computes x^L in $O(n^2)$ time.

7.5. Algorithm for Obtaining x^U

This case is similar to x^L. For the solution x^U defined at the end of Section 7.2 corresponding to constraint (7.10), let i_U be defined by

$$f_{i_U}(x^U_{i_U}) = \max_{1\le i\le n} f_i(x^L_i). \tag{7.20}$$

Lemma 7.10 There exists an x^U such that $x^U_{i_U} = x^b_{i_U}$ or $x^b_{i_U} + 1$.

Proof. Similar to Lemma 7.8. □

For $k = 1, 2, \cdots, n$, denote by $x^U(k, \alpha)$ a feasible solution of R, which minimizes $z(x)$ subject to

$$x_k = x^b_k + \alpha, \tag{7.21}$$

$$x_i \le x^b_i + 1 \text{ and } f_i(x_i) \le f_k(x_k) \text{ for all } i. \tag{7.22}$$

If $x^U(k, \alpha)$ does not exist, $z(x^U(k, \alpha))$ is set to ∞. Let $x^U(k)$ be one of the $x^U(k, 0)$ and $x^U(k, 1)$ satisfying

$$z(x^U(k)) = \min\{z(x^U(k, 0)), z(x^U(k, 1))\}. \tag{7.23}$$

Then x^U is equal to one of the $x^U(k)$ such that

$$z(x^U) = \min_{1\le k\le n} z(x^U(k)). \tag{7.24}$$

$x^U(k)$ can be computed as follows.

(i) For each $k = 1, 2, \cdots, n$ and $\alpha = 0, 1$, compute $x^U(k, \alpha)$

as follows. Let $\bar{x}_k = x_k^b + \alpha$ and $\bar{x}_i$ for $i \neq k$ be the maximum x_i satisfying (7.22). If $\sum_{i=1}^{n} \bar{x}_i < N$, $x^U(k, \alpha)$ does not exist ($z(x^U(k, \alpha))$ is set to ∞). Otherwise, using $u_i = \bar{x}_i$ for all i and $\ell_k = \bar{x}_k$ (other ℓ_i, $i \neq k$, do not change), $x^U(k, \alpha)$ is obtained by solving R_a.

(ii) Compute $x^U(k)$ and x^U by (7.23) and (7.24).

The entire algorithm is denoted XU.

Lemma 7.11 Algorithm XU computes x^U in $O(n^2)$ time.

7.6. Algorithm for Obtaining x*

As a result of the preceding discussion, an optimal solution x^* of R is now obtained by the following algorithm.

Algorithm OPTR

Input: N, n, $f_1, f_2, \cdots, f_n$, g, $\ell_1, \ell_2, \cdots, \ell_n$, $u_1, u_2, \cdots, u_n$.

Output: An optimal solution $x^* = (x_1^*, \cdots, x_n^*)$ of R.

Step 1 Compute x^a and x^b by Algorithm MAXIMIN and MINIMAX respectively. If $x^a = x^b$, $x^* \leftarrow x^a$ and stop.

Step 2 Call XL and XU to obtain x^L and x^U. Let x^* be equal to one of the x^L and x^U such that

$$z(x^*) = \min[z(x^L), z(x^U)].$$

x^* is an optimal solution of R. □

Theorem 7.12 Algorithm OPTR correctly computes an optimal solution x^* of R in $O(n^2 + n \log N)$ time (assuming that each evaluation of $f_i(x_i)$ is done in constant time).

Proof. The correctness immediately follows from the previous discussion. Since x^a and x^b are obtained in $O(n \log N)$ time by Lemma 7.7, and x^L and x^U are obtained in $O(n^2)$ time as discussed previously, the total time is $O(n^2 + n \log N)$. □

7.7 Conclusion

This chapter has proposed an efficient algorithm for the equipollent resource allocation problem, which is to minimize the maximum difference (measured by function g) of the resulting profits between each pair of activities. The proposed algorithm has been developed on the basis of the algorithm for the two simpler problems introduced in Section 7.2. One particular application of the equipollent problem is the apportionment problem, which will be further studied in the next chapter.

CHAPTER 8

AN EQUIPOLLENT RESOURCE ALLOCATION PROBLEM WITH APPLICATION TO OPTIMAL APPORTIONMENT

This chapter also deals with a minimax resource allocation problem discussed in the previous chapter. This chapter proposes another efficient algorithm for this problem. The algorithm requires $O(n^2 + \bar{T})$ running time, where n is the number of activities and $\bar{T}$ is the computation time to solve a continuous version of the problem. It is shown that this algorithm is advantageous over the one proposed in the previous chapter for a certain class of the problem. Then, this result is applied to the apportionment problem belonging to this class, and some computational results are reported.

8.1 Introduction

This chapter again considers the problem R discussed in the previous chapter. This is expressed as follows.

$$\text{R:}\quad \text{minimize } z(x) = g(\max_{1\le i\le n} f_i(x_i), \min_{1\le i\le n} f_i(x_i)) \tag{8.1}$$

$$\text{subject to } \sum_{i=1}^{n} x_i = N, \quad x_i\text{: nonnegative integers},$$

$$\ell_i \le x_i \le u_i, \quad i = 1,2,\cdots,n,$$

where f_i's are nondecreasing functions defined over $[\ell_i, u_i]$,

$g(\xi, \eta)$ is a nondecreasing (resp. nonincreasing) function in ξ (resp. η), N is a given positive integer, and ℓ_i and u_i are respectively the lower and upper bounds of x_i satisfying

$$\sum_{i=1}^{n} \ell_i \leq N \leq \sum_{i=1}^{n} u_i, \tag{8.2}$$

$$0 \leq \ell_i \leq u_i \leq N.$$

As shown in the previous chapter, this problem often arises whenever it is necessary to distribute a given amount of resources to a given set of activities so as to minimize the maximum difference of the resulting profits (i.e., $f_i(x_i)$'s) between activities. Chapter 7 has presented an efficient algorithm for the problem R, which requires $O(n^2 + n\log N)$ time.

This chapter presents another efficient algorithm for the problem R, which requires $O(n^2 + \bar{T})$ time, assuming that each evaluation of $f_i(x_i)$ and $f_i^{-1}(y_i)$ is done in constant time, where $\bar{T}$ is the time to solve the continuous version $\bar{R}$ obtained from R by dropping the integrality condition on x_i's. The idea of the algorithm is as follows. First, some relations between $\bar{R}$ and R are discussed. Then, based on the relations, an algorithm solving R is developed. This algorithm is shown to be faster than the one proposed in the previous chapter for a certain class of the problem.

This chapter applies this result to the apportionment problem belonging to this class. This problem is to distribute

a given number of seats to electoral districts so that the number of seats assigned to each state is as proportional to the population of the state as possible. A similar problem also occurs when a given number of seats are to be proportionally distributed among political parties according to their respective vote totals.

The apportionment problem has a long history in the United States (e.g., see [B1, B2, B3] and references cited therein), and several methods (such as the Huntington method and the quota method) have been proposed. The United States specifies a rule to determine the apportionment by a law, and the apportionment is updated from time to time corresponding to the population change. Some criteria, which an apportionment should satisfy, have been considered; the house monotone property, the quota property and certain consistency properties. In particular, the house monotone property, which requires that the number of seats given to a state does not decrease when the total number of seats is increased, has played a crucial role in developing the Huntington method and the quota method.

On the other hand, some countries specify the apportionment itself (not the rule) by their law. Since the apportionemnt is not updated so often under this circumstance, the house monotone property seems to be less important. Probably the impartiality measured by a certain criterion is of primary concern. For example, in Japan, the impartiality of the value of one vote has been

an important political issue, where the value of a vote in district i is measured by x_i/p_i, where x_i is the number of seats given to district i and p_i is its population. Several actions against the present apportionment has been taken and some are being put on trial. The current apportionment for the House of Representatives has the ratio $r \triangleq (\max_{1\le i\le n} x_i/p_i) / (\min_{1\le i\le n} x_i/p_i) = 3.5$. To judge whether this value of r is acceptable or not, it would be necessary to know the minimum possible value of r for a given set of sizes $p_i (i = 1,2,\cdots,n)$ and N.

This minimization problem (also under other criteria) is a special case of the equipollent resource allocation problem R, if $f_i(x_i)$ is interpreted as x_i/p_i. This chapter introduces five reasonable criteria for evaluation the impartiality and reports some computational results. Note that these five problems can be solved in $O(n^2)$ time by applying the algorithm presented in this chapter since the corresponding continuous versions can be solved in O(n) running time. This fact implies that the proposed algorithm is advantageous over the one in the previous chapter for the apportionment problem.

It is also shown by examples that none of apportionments under these criteria generally have the house monotone property, and that neither the Huntington method nor the quota method necessarily provides an optimal solution in this sense; the minimization of one of the criteria and the house monotone property

are not compatible.

This chapter is organized as follows. Section 8.2 describes some properties relating an optimal solution of $\bar{R}$ (i.e., the continuous version of R) to that of R. Sections 8.3 - 8.5 present an algorithm for obtaining an optimal solution of R. Section 8.6 gives an application to the apportionment problem.

8.2 Properties of Optimal Solutions

Some properties of optimal solutions of R are discussed in this section. These will be used in the subsequent sections to construct an algorithm for R.

Let $\bar{R}$ be the problem obtained from R by dropping the integrality condition on x_i's. In this case, each f_i is assumed to be a nondecreasing continuous function. The following maximin allocation problem with continuous variables is closely related to $\bar{R}$.

$$\bar{Q}: \quad \text{maximize} \quad \min_i f_i(x_i)$$

$$\text{subject to} \quad \sum_{i=1}^{n} x_i = N$$

$$\ell_i \le x_i \le u_i, \quad i = 1, 2, \ldots, n.$$

It is no difficult to show that an optimal solution of this problem is also optimal to $\bar{R}$ (unfortunately, this problem cannot be extended to integer solutions). Brown [B7] proposes an algorithm to solve $\bar{Q}$ (and hence $\bar{R}$) by using the following lemma.

Lemma 8.1 [B7] A feasible solution $\bar{x} = (\bar{x}_1, \bar{x}_2, \ldots, \bar{x}_n)$ satisfying the conditions $\sum_{i=1}^{n} \bar{x}_i = N$ and $\ell_i \le x_i \le u_i$, $i = 1, 2, \ldots, n$, is an optimal solution of $\bar{R}$, if there is a $\bar{\lambda}$ satisfying

$$\bar{x}_i = \begin{cases} \ell_i & \text{if } \bar{\lambda} < f_i(\ell_i) \\ f_i^{-1}(\bar{\lambda}) & \text{if } f_i(\ell_i) \le \bar{\lambda} \le f_i(u_i) \\ u_i & \text{if } \bar{\lambda} > f_i(u_i). \end{cases} \tag{8.3}$$

Moreover, there exists an optimal solution $\bar{x}$ of $\bar{R}$ satisfying condition (8.3). (The second condition of (8.3) implies $f_i(\bar{x}_i) = \bar{\lambda}$; such $\bar{x}_i$ may not be unique.) □

Brown's algorithm [B7] may be generally used to obtain $\bar{x}$ and $\bar{\lambda}$ of Lemma 8.1. If f_i's are specially structured, however, much simpler algorithm may exist. The apportionment problem discussed in Section 8.6 is such an example. In the subsequent discussion, we assume that an optimal solution $\bar{x}$ and an optimal Lagrange multiplier $\bar{\lambda}$ satisfying (8.3) are already obtained. Let I, J and K be defined by

$$I = \{i \mid f_i(\ell_i) \le \bar{\lambda} \le f_i(u_i), \quad 1 \le i \le n\}, \tag{8.4}$$

$$J = \{j \mid \bar{\lambda} < f_j(\ell_j), \quad 1 \le j \le n\}, \tag{8.5}$$

$$K = \{k \mid \bar{\lambda} > f_k(u_k), \quad 1 \le k \le n\}, \tag{8.6}$$

Let x^* be an optimal solution of R. Assume that $\bar{x}$ is not an integer solution, since otherwise $x^* = \bar{x}$ is obvious.

Lemma 8.2 There exists an optimal solution x^* satisfying at least one of the following inequalities.

(a) $x_i^* \geq \lfloor \bar{x}_i \rfloor$ for all i, or

(b) $x_i^* \leq \lceil \bar{x}_i \rceil$ for all i,

where $\lfloor x \rfloor$ ($\lceil x \rceil$) is the smallest (largest) integer not smaller (larger) than x.

Proof. If this is not the case, there exist k and ℓ with $x_k^* < \lfloor \bar{x}_k \rfloor$ and $x_\ell^* > \lceil \bar{x}_\ell \rceil$. Let I, J and K be those defined in (8.4) - (8.6). $k \notin J$ and $\ell \notin K$ are obvious since $\bar{x}_k = \ell_k$ and $\bar{x}_\ell = u_\ell$. Consider the feasible solution $\hat{x}$ defined by $\hat{x}_k = x_k^* + 1$, $\hat{x}_\ell = x_\ell^* - 1$ and $\hat{x}_i = x_i^*$ $(i \neq k, \ell)$. Then

$$\max_{1 \leq i \leq n} f_i(\hat{x}_i) \leq \max_{1 \leq i \leq n} f_i(x_i^*)$$

$$\min_{1 \leq i \leq n} f_i(\hat{x}_i) \geq \min_{1 \leq i \leq n} f_i(x_i^*)$$

hold because

$$f_k(x_k^*) \leq f_k(\hat{x}_k) \leq f_k(\lfloor \bar{x}_k \rfloor) \leq f_k(\bar{x}_k)$$

$$\leq f_\ell(\bar{x}_\ell) \quad (\text{since } k \in I \cup K \text{ and } \ell \in J \cup I)$$

$$\leq f_\ell(\lceil \bar{x}_\ell \rceil) \leq f_\ell(\hat{x}_\ell) \leq f_\ell(x_\ell^*).$$

Thus $z(\hat{x}) \leq z(x^*)$ (z is defined in (8.1)) follows. Repeating this, the lemma will be eventually proved. □

Lemma 8.2 suggests the following procedure to obtain an optimal solution x*.

(i) First compute an optimal solution $x^{\bar{L}}$ of the following problem.

$$R^{\bar{L}}: \quad \text{minimize} \quad z(x)$$
$$\text{subject to constraint of R,} \tag{8.7}$$
$$x_i \geq \lfloor \bar{x}_i \rfloor, \quad i = 1, 2, \ldots, n.$$

(ii) Secondly compute an optimal solution $x^{\bar{U}}$ of the following problem.

$$R^{\bar{U}}: \quad \text{minimize} \quad z(x)$$
$$\text{subject to constraint of R,} \tag{8.8}$$
$$x_i \leq \lceil \bar{x}_i \rceil, \quad i = 1, 2, \ldots, n.$$

(iii) Finally an optimal solution x* is chosen from $x^{\bar{L}}$ and $x^{\bar{U}}$ by

$$z(x^*) = \min \{z(x^{\bar{L}}), z(x^{\bar{U}})\}.$$

The following three sections show that this approach is in fact possible.

8.3. Algorithm for Obtaining $x^{\bar{L}}$

We start with the next lemma.

Lemma 8.3 Let $\bar{x}$ be an optimal solution of $\bar{R}$. Then the solution $x^{\bar{L}}$ (defined at the end of Section 8.2) satisfies

$$f_{i*}(x^{\bar{L}}_{i*}) = f_{i*}(\lfloor \bar{x}_{i*} \rfloor)$$

for some i* with

$$f_{i*}(x^{\bar{L}}_{i*}) = \min_{1 \le i \le n} f_i(x^{\bar{L}}_i).$$

Proof. Let I, J and K be defined as in (8.4)-(8.6). If $I = \phi$, $x^{\bar{L}} = \bar{x}$ since $\bar{x}$ is an integer solution. In this case, the lemma is obvious. If $I \neq \phi$, first observe that $f_k(x^{\bar{L}}_k) = f_k(\lfloor \bar{x}_k \rfloor) = f_k(u_k) < f_j(\ell_j) = f_j(\bar{x}_j) = f_j(\lfloor \bar{x}_j \rfloor) \le f_j(x^{\bar{L}}_j)$ holds for any $k \in K$ and $j \in J$ by Lemma 8.1. Furthermore $x^{\bar{L}}_{i'} = \lfloor \bar{x}_{i'} \rfloor$ holds for some $i' \in I$, since otherwise

$$\sum_{i=1}^{n} x^{\bar{L}}_i \ge \sum_{i \in I} (\lfloor \bar{x}_i \rfloor + 1) + \sum_{k \in K} \lfloor \bar{x}_k \rfloor + \sum_{j \in J} \lfloor \bar{x}_j \rfloor > \sum_{i=1}^{n} \bar{x}_i = N,$$

a contradiction to the feasibility of $x^{\bar{L}}$. Since $f_{i_1}(\lfloor \bar{x}_{i_1} \rfloor + 1) \ge \bar{\lambda} \ge f_{i_2}(\lfloor \bar{x}_{i_2} \rfloor)$ for any $i_1, i_2 \in I$, and $f_j(x^{\bar{L}}_j) \ge f_j(\lfloor \bar{x}_j \rfloor) = f_j(\ell_j) > f_i(\lfloor \bar{x}_i \rfloor)$ for any $j \in J$ and $i \in I$ by Lemma 8.1, the index i* satisfying $f_{i*}(x^{\bar{L}}_{i*}) = \min f_i(x^{\bar{L}}_i)$ belongs to either K

or I. In either case, $f_{i*}(x^{\bar{L}}_{i*}) = f_{i*}(\lfloor \bar{x}_{i*} \rfloor)$ is obvious from the above discussion. □

Now for each k with $1 \le k \le n$, consider the following problem

$$
\begin{aligned}
R^{\bar{L}}(k): \quad & \text{minimize} \quad z(x) = g(\max_i f_i(x_i), \min_i f_i(x_i)) \\
& \text{subject to constraint of R,} \\
& x_k = \lfloor \bar{x}_k \rfloor, \\
& \lfloor \bar{x}_i \rfloor \le x_i \le u_i \quad \text{and} \quad f_k(\lfloor \bar{x}_k \rfloor) \le f_i(x_i) \\
& \qquad\qquad \text{for } i = 1, 2, \ldots, n,
\end{aligned}
\tag{8.9}
$$

and denote an optimal solution by $x^{\bar{L}}(k)$. By Lemma 8.3, $x^{\bar{L}}$ is given as the $x^{\bar{L}}(k)$ satisfying

$$
z(x^{\bar{L}}) = \min_{1 \le k \le n} z(x^{\bar{L}}(k)). \tag{8.10}
$$

If $R^{\bar{L}}(k)$ does not have a feasible solution, $z(x^{\bar{L}}(k))$ is set to ∞.

By the assumption on g, $R^{\bar{L}}(k)$ is equivalent to

$$
\begin{aligned}
& \text{minimize} \quad \max_i f_i(x_i) \\
& \text{subject to constraint of } R^{\bar{L}}(k).
\end{aligned}
$$

A few algorithms for this minimax problem are known [B7, J1, F4]. Especially, [F4] contains the most efficient algorithm requiring $O(n \log M)$ time, if $M \ge n$, and $O(n)$ time otherwise, where $M \triangleq$

$N - \sum_{i=1}^{n} \lfloor \bar{x}_i \rfloor$. ([K14] proposes a different approach for solving the equipollent resource allocation pronlem R. The above algorithm is based on the result given by Frederickson and Johnson [F4].) Since $M = N - \sum_{i=1}^{n} \lfloor \bar{x}_i \rfloor = \sum_{i=1}^{n} \bar{x}_i - \sum_{i=1}^{n} \lfloor \bar{x}_i \rfloor \leq \sum_{i=1}^{n} (\bar{x}_i - \lfloor \bar{x}_i \rfloor) < n$, it follows that an optimal solution $x^{\bar{L}}(k)$ of $R^{\bar{L}}(k)$ can be obtained in $O(n)$ time. Let $X\bar{L}(k)$ denote the subroutine to obtain $x^{\bar{L}}(k)$ by following the procedure outlined in the above.

Lemma 8.4 Subroutine $X\bar{L}(k)$ correctly computes $x^{\bar{L}}(k)$ in $O(n)$ time if $x^{\bar{L}}(k)$ exists. If $x^{\bar{L}}(k)$ does not exist, $z(x^{\bar{L}}(k)) = \infty$ is output. □

To obtain $x^{\bar{L}}$ by (8.10), we need obtain $x^{\bar{L}}(k)$ for all k. The total time for this computation is $O(n^2)$.

8.4. Algorithm for Obtaining $x^{\bar{U}}$

This case can be treated in parallel to $x^{\bar{L}}$. The following lemma is proved in a way similar to Lemma 8.3.

Lemma 8.5 Let $\bar{x}$ be an optimal solution of $\bar{R}$. Then the solution $x^{\bar{U}}$ defined at the end of Section 8.2 satisfies

$$f_{i*}(x^{\bar{U}}_{i*}) = f_{i*}(\lceil \bar{x}_{i*} \rceil)$$

for any i* with

$$f_{i*}(x^{\bar{U}}_{i*}) = \max_i f_i(x^{\bar{U}}_i). \qquad \square$$

For k = 1, 2,..., n, let $x^{\bar{U}}(k)$ denote an optimal solution of

$$\begin{aligned} R^{\bar{U}}(k): \quad & \text{minimize} \quad z(x) = g(\max_i f_i(x_i), \min_i f_i(x_i)) \\ & \text{subject to constraint of } R, \\ & x_k = \lceil \bar{x}_k \rceil, \qquad (8.11) \\ & \ell_i \le x_i \le \lceil \bar{x}_i \rceil \text{ and } f_i(x_i) \le f_k(\lceil \bar{x}_k \rceil) \\ & \qquad\qquad \text{for } i = 1, 2, \ldots, n. \end{aligned}$$

Then $x^{\bar{U}}$ is given as the $x^{\bar{U}}(k)$ satisfying

$$z(x^{\bar{U}}) = \min_{1 \le k \le n} z(x^{\bar{U}}(k)). \qquad (8.12)$$

By the assumption on g, $R^{\bar{U}}(k)$ is equivalent to

$$\text{maximize} \quad \min_i f_i(x_i)$$

subject to constraint of $R^{\bar{U}}(k)$.

An optimal solution $x^{\bar{U}}(k)$ of $R^{\bar{U}}(k)$ can also be obtained in $O(n)$ time. The subroutine to obtain $x^{\bar{U}}(k)$ is denoted by $X\bar{U}(k)$.

Lemma 8.6 Subroutine $X\bar{U}(k)$ correctly computes $x^{\bar{U}}(k)$ in $O(n)$ time if $x^{\bar{U}}(k)$ exists. If $x^{\bar{U}}(k)$ does not exist, $z(x^{\bar{U}}(k)) = \infty$ is output. □

Consequently, $x^{\bar{U}}$ of (8.12) is obtained in $O(n^2)$ time.

8.5 Algorithm EQR for Obtaining x*

An optimal solution x* of R is now obtained by the following algorithm.

Algorithm EQR

Input: Problem R of (8.1).

Output: An optimal solution $x^* = (x_1^*, x_2^*, \ldots, x_n^*)$ of R.

Step 1 Compute an optimal solution $\bar{x}$ of $\bar{R}$ (the continuous version of R) by an appropriate algorithm. If $\bar{x}$ is an integer solution, $x^* \leftarrow \bar{x}$ and stop. Otherwise go to step 2.

Step 2 Call $\bar{XL}(k)$ for k = 1, 2,..., n to obtain $x^{\bar{L}}(k)$, and call $\bar{XU}(k)$ for k = 1, 2,..., n to obtain $x^{\bar{U}}(k)$. Choose $x^{\bar{L}}$ and $x^{\bar{U}}$ from $\{x^{\bar{L}}(k)\}$ and $\{x^{\bar{U}}(k)\}$ respectively so that

$$z(x^{\bar{L}}) = \min_k z(x^{\bar{L}}(k))$$

$$z(x^{\bar{U}}) = \min_k z(x^{\bar{U}}(k))$$

hold. Finally choose x* from $\{x^{\bar{L}}, x^{\bar{U}}\}$ to satisfy

$$z(x^*) = \min [\, z(x^{\bar{L}}), z(x^{\bar{U}}) \,].$$

Halt. x* is an optimal solution of R. □

Theorem 8.7 Algorithm EQR correctly computes an optimal solution x* of R in $O(n^2 + \bar{T})$ time, where $\bar{T}$ denotes the time to solve $\bar{R}$.

Proof. The correctness immediately follows from the previous discussion. Since $\bar{x}$ is obtained in $O(\bar{T})$ time, and $x^{\bar{L}}$ and $x^{\bar{U}}$ can be obtained in $O(n^2)$ time as discussed previously, the total time is $O(n^2 + \bar{T})$. □

8.6. Application to the Apportionment Problem

As discussed in Section 8.1, the apportionment problem may be viewed as an equipollent resource allocation problem discussed so far. Let $p = (p_1, \ldots, p_n)$ with $p_i > 0$ denote the numbers of voters in n electoral districts and N be a positive integer representing the total number of seats. An apportionment is an n-dimensional nonnegative integer vector $(x_1, \ldots, x_n)$ satisfying $\sum_{i=1}^{n} x_i = N$. Due to the integrality condition on x_i's, an apportionment satisfying $x_1/p_1 = x_2/p_2 = \cdots = x_n/p_n$ is not always possible. A number of apportionment algorithms are known, representing various view points of "best" apportionment. The house monotone property mentioned in Section 8.1 is one of the important qualifications of a good apportionment.

The Huntington method, extensively studied in [B2], is a method having the house monotone property. Corresponding to rank functions, used to carry out the computation, five types of Huntington methods, SD, HM, EP, W and J, are known. The quota method [B3], a variation of the Huntington method, satisfies the house monotone property and also the quota constraint:

$$\lfloor \bar{x}_i \rfloor \leq x_i \leq \lceil \bar{x}_i \rceil, \qquad i = 1, 2, \ldots, n, \tag{8.13}$$

where $\bar{x}$ is the continuous optimal apportionment satisfying $\bar{x}_1/p_1 = \bar{x}_2/p_2 = \cdots = \bar{x}_n/p_n$.

As mentioned in Section 8.1, the house monotone property may not be the primary concern under certain circumstances. Instead, we may want to optimize a certain measure representing the degree of impartiality of the apportionment. In some cases, this leads to the equipollent resource allocation problem discussed so far, i.e.,

$$\text{R:}\quad \text{minimize}\quad z(x) = g(A_{max}, A_{min}) \tag{8.14}$$

$$\text{subject to}\quad \sum_{i=1}^{n} x_i = N$$

$$\ell_i \le x_i \le u_i,\quad x_i\text{: nonnegative integers},\quad i = 1, 2, \ldots, n,$$

where $A_{max} = \max_i x_i/p_i$ and $A_{min} = \min_i x_i/p_i$, and g is nondecreasing in A_{max} and nonincreasing in A_{min}.

The following g's may be typical.

$$z_1(x) = g_1(A_{max}, A_{min}) = A_{max} - A_{min}, \tag{8.15}$$

$$z_2(x) = g_2(A_{max}, A_{min}) = \max[(A_{max} - b), (b - A_{min})]$$

$$= \max_i |x_i/p_i - b|, \tag{8.16}$$

$$z_3(x) = g_3(A_{max}, A_{min}) = A_{max}/A_{min}, \tag{8.17}$$

$$z_4(x) = g_4(A_{max}, A_{min}) = 1/A_{min} - 1/A_{max}, \tag{8.18}$$

$$z_5(x) = g_5(A_{max}, A_{min}) = \max[(1/A_{min} - 1/b), (1/b - 1/A_{max})]$$

$$= \max_i |p_i/x_i - 1/b|, \tag{8.19}$$

where $b = N/\sum_{i=1}^{n} p_i$.

Based on Lemma 8.1, an optimal solution $\bar{x}$ of the continuous version $\bar{R}$ of the above problems can be easily computed by the following algorithm.

Algorithm CONT

Step 1 Rearrange the elements of set $\cup_{i=1}^{n}\{\ell_i/p_i, u_i/p_i\}$ in the increasing order, and denote the resulting set by $\{y_1, y_2,\ldots,y_{2n}\}$ $(y_1 \le y_2 \le \ldots \le y_{2n})$.

Step 2 For a positive number y, define $f_i^{-1}(y)$ by

$$f_i^{-1}(y) = \begin{cases} u_i & \text{if } y > u_i/p_i \\ yp_i & \text{if } \ell_i/p_i \le y \le u_i/p_i \\ \ell_i & \text{if } y < \ell_i/p_i. \end{cases}$$

Obtain the interval $[y_j, y_{j+1}]$ such that $\sum_i f_i^{-1}(y_j) \le N \le \sum_i f_i^{-1}(y_{j+1})$ by binary search over the set $\{y_1, y_2,\ldots, y_{2n}\}$.

Step 3 (Compute $\bar{x}$) Let $J \leftarrow \{i \mid f_i^{-1}(y_{j+1}) = \ell_i\}$ and $K \leftarrow \{i \mid f_i^{-1}(y_j) = u_i\}$. Let $\bar{x}_i \leftarrow \ell_i$ for all $i \in J$ and $\bar{x}_i \leftarrow u_i$ for all $i \in K$. Let $I \leftarrow \{1, 2,\ldots, n\} - (J \cup K)$ and $N' \leftarrow N - \sum_{i \in J \cup K} \bar{x}_i$. Let $\bar{x}_i \leftarrow N'p_i / \sum_{i \in I} p_i$ for all $i \in I$. □

Lemma 8.8 Algorithm CONT correctly obtains an optimal solution $\bar{x}$ of $\bar{R}$ in $O(n \log n)$ time for problem R of (8.14).

Proof. The correctness is obvious from Lemma 8.1 (note that $\bar{\lambda} = N'/\sum_{i\in I} p_i$ is used). Next, computational time is analyzed. Step 1 requires $O(n \log n)$ time if an appropriate sorting algorithm (i.e., heap sort [A1]) is applied. Step 2 is repeated at most $O(\log n)$ time since the binary search is applied to the set $\{y_1, y_2, \ldots, y_{2n}\}$. Each iteration requires $O(n)$ time. Hence Step 2 requires $O(n \log n)$ time in total. Step 3 requires $O(n)$ time. Thus the total required time is $O(n \log n)$. □

Theorem 8.9 The apportionment problem under five criteria $g_1 - g_5$ can be solved in $O(n^2)$ time.

Proof. Obvious from Theorem 8.7 and Lemma 8.8. □

Next, we give some computational results. Using the above five criteria, we have computed optimal apportionment for the House of Councilors in Japan. As upper and lower bounds on x_i, the following cases are considered.

Case 1: $\ell_i = 0$ and $u_i = N$.

Case 2: $\ell_i = 1$ and $u_i = N$.

Case 3: $\ell_i = \lfloor \bar{x}_i \rfloor$ and $u_i = \lceil \bar{x}_i \rceil$ (i.e., the quota constraint of (8.13)).

Case 4: $\ell_i = \max(1, \lfloor \bar{x}_i \rfloor)$ and $u_i = \lceil \bar{x}_i \rceil$.

The computational results are listed in Table 8.1. Table 8.1

Table 8.1 Optimal apportionments in the House of Councilors in Japan. p_i is the number of voters in district i in 1976, and N = 76.

district	$\frac{p_i N}{\Sigma p_i}$	current apportionment	Huntington and Quota method									Our method					
			Case 1			Cases 1 & 2			Case 2			Case(s) 1	1 & 2	2	Case(s) 3	3 & 4	4
			W	J	Q	SD	HM	EP	W	J	Q	g_1-g_2	g_3-g_5	g_1-g_2	g_1-g_2	g_3-g_5	g_1-g_2
Hokkaido	3.6	4	3	4	4	3	3	3	3	3	3	4	3	3	4	3	3
Aomori	1.0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
Iwate	0.9	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
Miyagi	1.3	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
Akita	0.9	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
Yamagata	0.9	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
Fukushima	1.3	2	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
Ibaragi	1.6	2	2	2	2	2	2	1	1	1	1	2	2	2	2	2	2
Tochigi	1.2	2	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
Gumma	1.2	2	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
Saitama	3.2	2	3	4	4	3	3	3	3	3	3	4	3	3	4	3	3
Chiba	2.8	2	3	3	3	3	2	3	3	3	3	3	3	3	3	2	2
Tokyo	7.9	4	7	10	8	6	7	7	7	8	8	10	6	6	8	7	7
Kanagawa	4.3	2	4	5	5	4	4	4	4	4	4	5	4	4	5	4	4
Niigata	1.6	2	2	2	2	2	2	2	1	1	1	2	2	2	2	2	2
Toyama	0.7	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
Ishikawa	0.7	1	1	0	1	1	1	1	1	1	1	0	1	1	1	1	1
Fukui	0.5	1	0	0	0	1	1	1	1	1	1	0	1	1	0	1	1
Yamanashi	0.5	1	1	0	0	1	1	1	1	1	1	0	1	1	0	1	1
Nagano	1.4	2	1	1	2	2	1	1	1	1	1	1	2	2	2	1	1
Gifu	1.3	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
Shizuoka	2.2	2	2	3	3	2	2	2	2	2	2	3	2	2	3	2	2
Aichi	3.9	3	4	5	4	3	3	3	4	4	4	5	3	3	4	3	3
Mie	1.1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
Shiga	0.7	1	1	0	0	1	1	1	1	1	1	0	1	1	0	1	1
Kyoto	1.7	2	2	2	2	2	2	2	2	1	1	2	2	2	2	2	2
Osaka	5.4	3	5	7	6	4	5	5	5	6	6	7	4	4	6	5	5
Hyogo	3.4	3	3	4	4	3	3	3	3	3	3	4	3	3	4	3	3

(Table continued)

Table 8.1

continued

district	$\frac{P_1 N}{\Sigma P_1}$	current apportionment	Huntington and Quota method									Our method					
			Case 1			Cases 1 & 2			Case 2			Case(s) 1	Case(s) 1 & 2	Case(s) 2	Case(s) 3	Case(s) 3 & 4	Case(s) 4
			W	J	Q	SD	HM	EP	W	J	Q	g_1-g_2	g_3-g_5	g_1-g_2	g_1-g_2	g_3-g_5	g_1-g_2
Nara	0.7	1	1	0	1	1	1	1	1	1	1	0	1	1	1	1	1
Wakayama	0.7	1	1	0	1	1	1	1	1	1	1	0	1	1	1	1	1
Tottori	0.4	1	0	0	0	1	1	1	1	1	1	0	1	1	0	1	1
Shimane	0.5	1	1	0	0	1	1	1	1	1	1	0	1	1	0	1	1
Okayama	1.3	2	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
Hiroshima	1.8	2	2	2	2	2	2	2	2	1	1	2	2	2	2	2	2
Yamaguchi	1.1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
Tokushima	0.6	1	1	0	0	1	1	1	1	1	1	0	1	1	0	1	1
Kagawa	0.7	1	1	0	0	1	1	1	1	1	1	0	1	1	0	1	1
Ehime	1.0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
Kochi	0.6	1	1	0	0	1	1	1	1	1	1	0	1	1	0	1	1
Fukuoka	3.0	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3
Saga	0.6	1	1	0	0	1	1	1	1	1	1	0	1	1	0	1	1
Nagasaki	1.0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
Kumamoto	1.2	2	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
Oita	0.8	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
Miyazaki	0.7	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
Kagoshima	1.2	2	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
Okinawa	0.6	1	1	0	0	1	1	1	1	1	1	0	1	1	0	1	1
value of g_3		5.28	—	—	—	3.31	3.43	3.92	4.01	4.09	4.09	—	3.31	—	—	3.43	—

also contains the present apportionment determined by a law, and the apportionments obtained by the Huntington method of types SD, HM, EP, W and J, and the quota method Q.

Table 8.1 shows that optimal apportionments under $g_1 - g_2$ (resp. $g_3 - g_5$) coincide in each of the Cases 1-4. Six types of the Huntington method are however mutually different in each of Cases 1 and 2 (Huntington methods are usually not applied to Cases 3 and 4 though theoretically possible). The current apportionment is quite different from any one of the optimal apportionments. For example, the current value of g_3 is 5.28, while the optimal value of g_3 is 3.31. Finally, the Huntington methods and minimization of $g_1 - g_5$ do not seem to be inherently related. In each of Cases 1 - 4, the result obtained by minimizing g_j coincides with the result obtained by one of the Huntington method. But such Huntington method varies according to the considered cases.

Finally we present several examples to examine the compatibility between the house monotone property possessed by the Huntington method and the optimality under one of $g_1 - g_5$. In these examples $\ell_i = 0$ and $u_i = N$ are assumed for i=1, 2,..., n.

Example 1: Let $n = 4$, $N = 18$, $p_1 = p_2 = 21$, $p_3 = 77$, $p_4 = 33$. Then $b = 0.118$ follows and $x^* = (2, 2, 10, 4)$ is the unique optimal solution under $g_1 - g_4$. Here $A_{max} = x_3^*/p_3 = 0.130$, $A_{min} = x_1^*/p_1 =$

$x_2^*/p_2 = 0.095$, $z_1(x^*) = 0.130 - 0.095 = 0.035$, $z_2(x^*) = 0.118 - 0.095 = 0.023$, $z_3(x^*) = 0.130/0.095 = 1.37$, and $z_4(x^*) = 1/0.095 - 1/0.130 = 2.8$. When N is increased to 19, it is easy to see that $b = 0.125$ and $x^* = (3, 3, 9, 4)$ is the unique optimal solution, where $A_{max} = x_1^*/p_1 = x_2^*/p_2 = 0.143$, $A_{min} = x_3^*/p_3 = 0.117$, $z_1(x^*) = 0.143 - 0.117 = 0.026$, $z_2(x^*) = 0.143 - 0.125 = 0.018$, $z_3(x^*) = 0.143/0.117 = 1.222$, and $z_4(x^*) = 1/0.117 - 1/0.143 = 1.56$. Thus x_3 decreases by one as a result of increasing N by one; the house monotone property is violated.

Example 2: Let $n = 4$, $N = 18$, $p_1 = p_2 = 21$, $p_3 = 84$, $p_4 = 36$. Then $b = 0.111$ holds and $x^* = (2, 2, 10, 4)$ is the unique optimal solution under g_5. Here $A_{max} = x_3^*/p_3 = 0.119$, $A_{min} = x_1^*/p_1 = x_2^*/p_2 = 0.095$, $z_5(x^*) = 1/0.095 - 1/0.111 = 1.5$. When N is increased to 19, $b = 0.117$ and $x^* = (3, 3, 9, 4)$ is the unique optimal solution, where $A_{max} = x_1^*/p_1 = x_2^*/p_2 = 0.143$, $A_{min} = x_4^*/p_4 = 0.111$, $z_5(x^*) = 1/0.117 - 1/0.143 = 1.526$. This also violates the house monotone property.

Example 3: Consider Example 1 with $N = 18$. Then W, EP, HM and SD of the Huntington method (see reference [B3] for details) provide solutions $x_W = x_{EP} = x_{HM} = (3, 2, 9, 4)$ and $x_{SD} = (3, 3, 8, 4)$ respectively. But an optimal solution in the sense of $g_1 - g_4$ is $x^* = (2, 2, 10, 4)$. Next consider Example 1 with $N = 19$. Then J of the Huntington method and the quota method Q provide

solutions $x_J = x_Q = (3, 2, 10, 4)$. But an optimal solution in the sense of $g_1 - g_4$ is $x^* = (3, 3, 9, 4)$.

Example 4: Consider Example 2 with $N = 18$. Then the SD method provides a solution $x_{SD} = (3, 3, 8, 4)$. But an optimal solution in the sense of g_5 is $x^* = (2, 2, 10, 4)$. Next consider Example 2 with $N = 19$. Then J, W, EP, HM of the Huntington method and the quota method provide solutions $x_J = (2, 2, 11, 4)$, $x_W = x_{EP} = x_{HM} = (3, 2, 10, 4)$ and $x_Q = (2, 2, 10, 5)$. But an optimal solution in the sense of g_5 is $x^* = (3, 3, 9, 4)$.

We can see from these examples that the optimality under $g_1 - g_5$ is not compatible with the house monotone property. However, as obvious from the discussion given so far, the quota property (8.13) and the optimality can simultaneously taken into account; such an apportionment is computed by Algorithm EQR if the lower and the upper bound constraints are set as in Case 3 or 4 of the above computational results.

8.7 Conclusion

This chapter has proposed an efficient algorithm for the equipollent resource allocation problem, which has been discussed in Chapter 7. The proposed algorithm requires $O(n^2 + \bar{T})$ running time, while the algorithm proposed in Chapter 7 requires $O(n^2 + n\log N)$ running time. This algorithm is superior to the previous one in Chapter 7 if a continuous version of the problem (defined in Section 8.1) is efficiently solved (i.e., $\bar{T} < n\log N$). This chapter has then applied this result to the apportionment problem, which has been shown to be solved in $O(n^2)$ time. Computational results have been also reported. Furthermore, it has been also shown that none of five criteria introduced in Section 8.6 is compatible with criteria of the existing methods such as the Huntington method and the quota method.

It should be mentioned that in many cases an optimal solution is not uniquely determined, since the value of the objective function is determined only by $\max_i f_i(x_i)$ and $\min_i f_i(x_i)$. Therefore, if necessary, some secondary criterion should be introduced, which is left for the future research.

CHAPTER 9

AN EFFICIENT ALGORITHM FOR K BEST SOLUTIONS FOR THE RESOURCE ALLOCATION PROBLEM

This chapter proposes an algorithm for obtaining K best solutions for the simple resource allocation problem discussed in Chapter 4. It requires $O(T^* + K\log K + K\sqrt{n\log n})$ time and $O(K\sqrt{n\log n} + n)$ space, where n is the number of variable and T^* is the computational time to obtain the first best solution. The idea of the algorithm is as follows. First, a subroutine is developed to efficiently find the second best solution in a certain subset of all feasible solutions, when the first best solution is already given. Secondly, a systematic partition scheme is developed not to repeatedly generate the same solution as the already obtained ones. Incorporating the above subroutine into the partition scheme, the algorithm partitions the solution space into small subsets step by step, and computes the first best solution, the second best solution, $\cdots$, the K-th best one by systematically obtaining the best solutions in the partitioned subsets.

9.1 Introduction

The simple resource allocation problem has been discussed in Chapter 4. It is noted, however, that the resource allocation problems arising in real world do not always have such a simple structure but usually have some complicated side constraints. References [S6, S7, W1] contain some examples of such resource allocation problems with more than one constraint. An optimal solution to any one of these complicated problems should be found among K best solutions to the simple resource allocation problems which are derived by neglecting the side constraints. In this respect, it is important to obtain more than one good solution, preferably K best solutions to the simple problem.

This chapter proposes an efficient algorithm for obtaining K best solutions to the simple resource allocation problem. It requires $O(T^* + K\log K + K\sqrt{n\log n})$ time and $O(K\sqrt{n\log n} + n)$ space, where T^* denotes the time to obtain the first best solution. It partitions the solution space into small subsets step by step, and computes the first best solution, the second best one, $\cdots$, the K-th best one by systematically obtaining the best solutions in the partitioned subsets. The partition scheme used in the algorithm is based on the framework developed in Chapters 2 and 3. Development of the systematic partition scheme and of the efficient subroutine for obtaining the best solutions in the partitioned subsets makes it possible to yield an efficient

algorithm.

This chapter is organized as follows. Section 9.2 gives necessary definitions and basic results. Section 9.3 gives an outline of the algorithm. Section 9.4 gives a detailed description of the algorithm, and analyzes the time and space requirement. The algorithm proposed in Sections 9.3 and 9.4 requires $O(T^* + K\log K + Kn)$ time and $O(Kn)$ space. These are respectively reduced to $O(T^* + K\log K + K\sqrt{n\log n})$ and $O(K\sqrt{n\log n} + n)$ in Section 9.5.

9.2 Definitions and Basic Concepts

In this section, we give some definitions and basic concepts. First, the simple resource allocation problem discussed in Chapter 4 is reviewed.

$$\text{P:}\quad \text{minimize } z(x) = \sum_{i=1}^{n} f_i(x_i)$$
$$\text{subject to } \sum_{i=1}^{n} x_i = N \text{ and } x_i\text{: nonnegative integers,}$$

where f_i's are convex functions defined over [0, N] and N is a positive integer. Let an integer solution satisfying $\sum_{i=1}^{n} x_i = N$ and $x_i \geq 0$ be called feasible. The k-th best solution $x^k=(x_1^k, x_2^k, \ldots, x_n^k)$ is defined recursively as follows.

(i) x^1 is an optimal solution of P, i.e., a feasible solution minimizing the objective value z(x).

(ii) x^k with $k \geq 2$ is a feasible solution of P with the minimum objective value among those different from $x^1, x^2, \ldots, x^{k-1}$.

For an integer x_i with $0 \leq x_i \leq N$, let

$$d_i^+(x_i) = f_i(x_i + 1) - f_i(x_i), \tag{9.1}$$

$$d_i^-(x_i) = f_i(x_i) - f_i(x_i - 1), \tag{9.2}$$

where $f_i(-1) = +\infty$ and $f_i(N+1) = +\infty$ are assumed by convention. Note that $d_i^+(x_i) = d_i^-(x_i + 1)$, and that $d_i^+(x_i)$ and $d_i^-(x_i)$ are both non-decreasing by the convexity of f_i. We assume throughout this paper that each $f_i(x_i)$ can be evaluated in constant time. For a feasible

solution $x = (x_1, x_2, \ldots, x_n)$, a pair of indices [i, j] is called an exchange if $0 \le x_i < N$, $0 < x_j \le N$ and $i \neq j$. Applying an exchange [i, j] to x yeilds another feasible solution $x' = (x_1, \ldots, x_i+1, \ldots, x_j-1, \ldots, x_n)$ with the objective value $z(x) + c(i, j)$, where

$$c(i, j) = d_i^+(x_i) - d_j^-(x_j).$$

Lemma 9.1 [M1] A feasible solution x is optimal if and only if there is no exchange with negative cost. □

Lemma 9.2 For an optimal solution x^1, let [i, j] be an exchange with the minimum cost (which is nonnegative). Then the solution $x^{\#} = (x_1^1, \ldots, x_i^1+1, \ldots, x_j^1-1, \ldots, x_n^1)$ obtained by applying [i, j] to x^1 is a second best solution x^2.

Proof. Let $\tilde{x}$ be a second best solution not equal to $x^{\#}$. Then there exists a pair of indices p, q such that $\tilde{x}_p > x_p^1$ and $\tilde{x}_q < x_q^1$. From $d_p^+(\tilde{x}_p - 1) \ge d_p^+(x_p^1)$ and $d_q^-(\tilde{x}_q+1) \le d_q^-(x_q^1)$, it follows that a feasible solution $x' = (\tilde{x}_1, \ldots, \tilde{x}_p - 1, \ldots, \tilde{x}_q+1, \ldots, \tilde{x}_n)$ has the objective value not grater than $\tilde{x}$ because $z(\tilde{x}) - z(x') = d_p^+(\tilde{x}_p-1) - d_q^-(\tilde{x}_q + 1) \ge d_p^+(x_p^1) - d_q^-(x_q^1) \ge d_i^+(x_i^1) - d_j^-(x_j^1) \ge 0$ by the optimality of x^1 and the minimality of exchange [i, j]. Repeating this, we eventually obtain a feasible solution $\hat{x}$ such that $z(\tilde{x}) \ge z(\hat{x})$, and $\hat{x}_\ell = x_\ell^1 + 1$ and $\hat{x}_m = x_m^1 - 1$ for some ℓ and m but $\hat{x}_k = x_k^1$ for all $k \neq \ell$, m. Then $(z(\tilde{x}) \ge) z(\hat{x}) \ge z(x^{\#})$ follows from $z(\hat{x}) - z(x^{\#}) = (d_\ell^+(x_\ell^1) - d_m^-(x_m^1)) - (d_i^+(x_i^1) - d_j^-(x_j^1)) \ge 0$ by the minimality of [i, j]. This implies $z(\tilde{x}) = z(x^{\#})$ and $x^{\#}$ is also a second best solution. □

To construct an algorithm for computing x^k for $k \geq 3$, we need generalize Lemma 9.2 as follows. The proof is similar to Lemma 9.2.

Lemma 9.3 Let two n-dimensional integer vectors $\bar{x}$ and $\underline{x}$ with $0 \leq \underline{x} \leq \bar{x}$ be given. Let $x = x^*$ be a best feasible solution of P among those satisfying

$$\underline{x}_i \leq x_i \leq \bar{x}_i, \quad i = 1, 2, \ldots, n.$$

Then a second best solution $\tilde{x}$ satisfying the above constraint is obtained by applying [i', j'] to x^*, where [i', j'] is a minimum exchange satisfying $x^*_{i'} < \bar{x}_{i'}$ and $x^*_{j'} > \underline{x}_{j'}$. □

9.3 The Outline of the Entire Algorithm

Our algorithm consists of routines COMPBS and KBS. COMPBS computes x^k when the first k-1 best solutions $x^1, x^2, \ldots, x^{k-1}$ are given. KBS generates all the K best solutions using COMPBS as a subroutine.

Now assume that $x^1, x^2, \ldots, x^{k-1}$ $(k > 1)$ have been generated. The set of remaining feasible solutions is partitioned into k-1 disjoint subsets:

$$P^h_{k-1} = \{x^\ell \mid \ell > k-1,\ \underline{x}^h(k-1) \le x^\ell \le \bar{x}^h(k-1)\}, \quad h = 1, 2, \ldots, k-1. \tag{9.3}$$

As will be discussed shortly, these sets have the property that x^k is equal to a solution with the minimum objective value among those obtained as best solutions in P^h_{k-1}'s respectively.

Vectors $\underline{x}^h(k-1)$ and $\bar{x}^h(k-1)$ are recursively defined as follows. Initially when $k = 2$ (i.e., only x^1 is obtained), $\bar{x}^h(1)$ and $\underline{x}^h(1)$ for $h = 1$ are given by

$$\begin{aligned} \bar{x}^1(1) &= (N, N, \ldots, N) \\ \underline{x}^1(1) &= (0, 0, \ldots, 0) \end{aligned} \tag{9.4}$$

In general, let $\bar{x}^h(k-1)$ and $\underline{x}^h(k-1)$ $(1 \le h \le k-1)$ be given, and assume that x^k is obtained from x^{h*} by applying an exchange $[i^*, j^*]$. Then $\bar{x}^h(k)$ and $\underline{x}^h(k)$ are defined as follows,

$$
\begin{aligned}
&\bar{x}^{h*}(k) = (\bar{x}_1^{h*}(k-1), \ldots, x_{i*}^{h*}, \ldots, \bar{x}_n^{h*}(k-1)),\\
&\underline{x}^{h*}(k) = \underline{x}^{h*}(k-1),\\
&\bar{x}^{k}(k) = \bar{x}^{h*}(k-1),\\
&\underline{x}^{k}(k) = (\underline{x}_1^{h*}(k-1), \ldots, x_{i*}^{k}(=\underline{x}_{i*}^{h*}+1), \ldots, \underline{x}_n^{h*}(k-1)),\\
&\bar{x}^{\ell}(k) = \bar{x}^{\ell}(k-1),\ \underline{x}^{\ell}(k) = \underline{x}^{\ell}(k-1) \text{ for all } \ell \neq h^*, k.
\end{aligned}
\tag{9.5}
$$

These new sets define P_k^h for $h = 1, 2, \ldots, k$. From this definition and Lemma 9.3, the next lemma is obvious.

Lemma 9.4 Let k be $2 \le k \le K$.

(1) For $h = 1, 2, \ldots, k-1$, x^h is a best feasible solution satisfying

$$\underline{x}^h(k-1) \le x \le \bar{x}^h(k-1). \tag{9.6}$$

Furthermore, no other x^ℓ (ℓ=1, 2, ..., h-1, h+1, ..., k-1) satisfy (9.6).

(2) Any feasible solution x of P satisfies (9.6) for exactly one h with $1 \le h \le k-1$. □

Lemma 9.4 (2) asserts that

$$\bigcup_{h=1}^{k-1} P_{k-1}^h = (\text{the set of all feasible solutions of P}) - \{x^1, x^2, \ldots, x^{k-1}\}$$

and $P_{k-1}^h \cap P_{k-1}^\ell = \phi$ for $h \neq \ell$, since the condition $\ell > k-1$ in each P_{k-1}^h excludes $x^1, x^2, \ldots, x^{k-1}$. Thus letting $\tilde{x}^h$ be a best feasible solution in each P_{k-1}^h, x^k is given as a best one in set $\{\tilde{x}^h \mid h=1, 2, \ldots, k-1\}$.

Lemma 9.4 (1) tells that $\tilde{x}^h$ is a second best solution among those satisfying (9.6).

In order to compute $\tilde{x}^h$ according to Lemma 9.3, we maintain two sets of labels (if $d_i^+(x_i^h) = d_{i'}^+(x_{i'}^h)$, both are stored)

$$\begin{aligned} D_+^h(k-1) &= \{d_i^+(x_i^h) \mid 1 \le i \le n,\ x_i^h < \bar{x}_i^h(k-1)\}, \\ D_-^h(k-1) &= \{d_j^-(x_j^h) \mid 1 \le j \le n,\ x_j^h > \underline{x}_j^h(k-1)\}, \end{aligned} \tag{9.7}$$

for $h = 1, 2, \ldots, k-1$. $D_+^h(k-1)$ and $D_-^h(k-1)$ contain at most $O(n)$ labels. A minimum exchange $[i', j']$ of Lemma 9.3 for $\bar{x} = \bar{x}^h(k-1)$ and $\underline{x} = \underline{x}^h(k-1)$ is then determined as follows. Let i_1 and i_2 be the indices of the first and the second minimum $d_i^+(x_i^h)$'s in $D_+^h(k-1)$, respectively, and j_1 and j_2 be the indices of the first and the second maximum $d_j^-(x_j^h)$'s in $D_-^h(k-1)$. Then

$$\begin{aligned} [i', j'] &= [i_1, j_1] \text{ if } i_1 \neq j_1 \\ &= [p, q] \text{ if } i_1 = j_1, \\ \text{where } c(p, q) &= \min[c(i_1, j_2), c(i_2, j_1)], \\ &\text{where } P \in \{i_1, i_2\},\ q \in \{j_1, j_2\}. \end{aligned} \tag{9.8}$$

Note that i_1, i_2, j_1 and j_2 are computed in $O(\log n)$ time if $D_{\pm}^h(k-1)$'s use appropriate data structure (any efficient priority queue discussed in [A1, K16] for example). $[i', j']$ is then computed by (9.8) and $\tilde{x}^h$ is obtained from x^h by applying exchange $[i', j']$.

When x^k is obtained as $\tilde{x}^{h*}$ (which is generated from x^{h*} by exchange $[i*, j*]$), $D_{\pm}^h(k)$'s are computed from $D_{\pm}^h(k-1)$'s as follows

(see (9.5)).

$$D_+^{h^*}(k) = D_+^{h^*}(k-1) - \{d_{i^*}^+(x_{i^*}^{h^*})\}, \tag{9.9}$$

$$D_-^{h^*}(k) = D_-^{h^*}(k-1), \tag{9.10}$$

$$D_+^{k}(k) = D_+^{h^*}(k-1) \cup \{d_{i^*}^+(x_{i^*}^{h^*}+1),\ d_{j^*}^+(x_{j^*}^{h^*}-1)\} - \{d_{i^*}^+(x_{i^*}^{h^*}),\ d_{j^*}^+(x_{j^*}^{h^*})\}, \tag{9.11}$$

$$D_-^{k}(k) = D_-^{h}(k-1) \cup \{d_{j^*}^-(x_{j^*}^{h^*}-1)\} - \{d_{i^*}^-(x_{i^*}^{h^*}),\ d_{j^*}^-(x_{j^*}^{h^*})\}, \tag{9.12}$$

$$D_+^{h}(k) = D_+^{h}(k-1),\ D_-^{h}(k) = D_-^{h}(k-1) \text{ for } h \neq h^*,\ k. \tag{9.13}$$

(Although not explicitly written, it should be understood in (9.11) and (9.12) that those $d_i^{\pm}(x_i^h)$'s whose variables violate the condition of (9.7) are not included in $D_{\pm}^k(k)$.) Using appropriate data structure of a priority queue, the deletion of an element from $D_{\pm}^h(k-1)$, and the addition of an element to $D_{\pm}^h(k-1)$ are respectively done in $O(\log n)$ steps (e.g., [A1, K16]). Thus $D_{\pm}^{h^*}(k)$ are computed in $O(\log n)$ steps and $D_{\pm}^k(k)$ are computed in $O(n)$ steps (since $D_{\pm}^{h^*}(k-1)$ must be copied and it requires $O(n)$ steps). Other $D_{\pm}^h(k)$'s do not require any computation time because $D_{\pm}^h(k-1)$ can be used as $D_{\pm}^h(k)$ with no change.

We note that each P_{k-1}^h is represented in our algorithm by the following list

$$P_{k-1}^h = (c', [i', j'], x^h, \underline{x}^h(k-1), \bar{x}^h(k-1), D_-^h(k-1), D_+^h(k-1)), \tag{9.14}$$

where $c' = z(x^h) + c(i', j')$ and $[i', j']$ is a minimum exchange with respect to $\bar{x}^h(k-1)$ and $\underline{x}^h(k-1)$. This list uses $O(n)$ space.

The computation of $x^k = \tilde{x}^{h*}$, $\underline{x}^h(k)$, $\bar{x}^h(k)$ and $D_{\pm}^h(k)$ for each k described above constitutes Subroutine COMPBS. COMPBS is repeated for $k = 2, 3, \ldots, K$. The entire procedure is organized as KBS.

9.4 Algorithm KBS and COMPBS

This section describes algorithms KBS and COMPBS in an ALGOL-like language, and then analyzes its running time.

```
Procedure KBS(P, K);
begin
   comment This procedure computes x^1, x^2, ..., x^K together
                with their costs c^1, c^2, ..., c^K. If P does
                not have K feasible solutions, KBS terminates
                after generating all feasible solutions;
1      Find an optimal solution x^1 for Problem P;
2      x̄^1(1) ← (N, N, ..., N); x̲^1(1) ← (0, 0, ..., 0); Compute D_^1(1)
       and D_+^1(1);
3      Find a minimum exchange [i', j'] with cost c(i', j');
4      P_1^1 ← (z(x^1)+c(i', j'), [i', j'], x^1, x̲^1(1), x̄^1(1), D_^1(1),
           D_+^1(1));
5      For k = 2 until K do call COMPBS(P_{k-1}^h | h=1, 2, ..., k-1);
end KBS;

Subroutine COMPBS(P_{k-1}^h | h = 1, 2, ..., k-1);
begin
1      Find P_{k-1}^{h*} = (c*, [i*, j*], x^{h*}, x̲^{h*}(k-1), x̄^{h*}(k-1), D_^{h*}(k-1),
           D_+^{h*}(k-1)) with the minimum cost c* among P_{k-1}^h, h = 1, 2,
           ..., k-1;
2      if c* = ∞ then stop (all feasible solutions have been output
```

and P has only k-1 feasible solutions);

3 else begin

comment Computation of x^k;

4 Output [i*, j*], h* and c* as x^k (x^k is obtained from x^{h*} by exchange [i*, j*]) and c^k respectively;

comment Updating P_k^{h*};

5 Construct $\underline{x}^{h*}(k)$ and $\bar{x}^{h*}(k)$ by (9.5);

6 Construct $D_-^{h*}(k)$ and $D_+^{h*}(k)$ by (9.9) and (9.10);

7 Using (9.8), find a minimum exchange [i', j'] for $D_{\pm}^{h*}(k)$;

8 if such [i', j'] exists then $P_k^{h*} \leftarrow (c^{h*} + c(i', j'), [i', j'], x^{h*}, \underline{x}^{h*}(k), \bar{x}^{h*}(k), D_-^{h*}(k), D_+^{h*}(k))$;

9 else (i.e., $D_+^{h*}(k) = \phi$ or $D_-^{h*}(k) = \phi$) $P_k^{h*} \leftarrow (\infty, \phi, \phi, \phi, \phi, \phi, \phi)$;

comment Computation of P_k^k;

10 Construct $\underline{x}^k(k)$ and $\bar{x}^k(k)$ by (9.5);

11 Construct $D_-^k(k)$ and $D_+^k(k)$ by (9.11) and (9.12);

12 Using (9.8), find a minimum exchange [i', j'] for $D_{\pm}^k(k)$;

13 if such [i', j'] exists then $P_k^k \leftarrow (c^k + c(i', j'), [i', j'], x^k, \underline{x}^k(k), \bar{x}^k(k), D_-^k(k), D_+^k(k))$;

14 else $P_k^k \leftarrow (\infty, \phi, \phi, \phi, \phi, \phi, \phi)$;

comment Computation of other P_k^h;

15 $P_k^h \leftarrow P_{k-1}^h$ for $h \neq h^*$, k;

16 end

return

end COMPBS;

Lemma 9.5 For each k = 2, 3, ..., K, COMPBS correctly computes x^k and P_k^h for h = 1, 2, ..., k in O(log K + n) time.

Proof. The correctness is obvious from the discussion in Sections 9.2 and 9.3. The time requirement is considered here. Line 1 of COMPBS is done in $O(\log(k-1)) \leq O(\log K)$ time if P_{k-1}^h's are linked in the form of an efficient priority queue [A1, K16]. Lines 2 and 3 are executed in constant time. Line 4 is executed in constant time since x^k is output only by h* and [i*, j*] instead of the full vector $x^k = (x_1^k, \ldots, x_n^k)$. Line 5 is executed in constant time as obvious from (9.5). Line 6 requires in O(log n) time as explained after (9.13). Lines 7 and 12 require O(log n) time as explained after (9.8). Hence P_k^{h*} of lines 8 and 9 is obtained in O(log n) time. Line 10 requires O(n) time as is obvious from (9.5) (note that O(n) is necessary to copy $\underline{x}^{h*}(k-1)$ and $\bar{x}^{h*}(k-1)$ beforehand). Line 11 requires O(n) time as explained after (9.13). This implies that P_k^k of line 13 or 14 is constructed in O(n) time. Line 15 requires constant time because it is accomplished by keeping the previous data. The adjustment of links among P_k^h (h = 1, 2, ..., k) (which are used at line 1) as a result of the addition of P_k^k and the modification of P_k^{h*} is done in $O(\log k)$ ($\leq O(\log K)$) time.

Thus all computation in COMPBS is done in $O(\log K+n)$ steps. □

Theorem 9.5 KBS correctly generates the K best solutions from x^1 to x^K in $O(T^*+Kn+K \log K)$ time and $O(Kn)$ space, where T^* is the time required to compute x^1. If P does not have K feasible solutions, KBS terminates after generating all feasible solutions.

Proof. The correctness of KBS follows from the previous discussion. The time complexity is analyzed here. Line 1 of KBS requires T^* time. Line 2 obviously requires $O(n)$ time since the initial construction of a priority queue is done in $O(n)$ time ([A1, K16]). Line 3 requires $O(\log n)$ time as explained after (9.8), and line 4 requires $O(n)$ time. Line 5 calls COMPBS K-1 times, and requires $O(K \log K+Kn)$ time in total by Lemma 9.5. Thus total time is $O(T^*+Kn+K \log K)$. Finally $O(Kn)$ space is required to store P_k^h $(h=1, 2, \ldots, k)$ since each P_k^h needs $O(n)$ space; the space for other data is obviously dominated by $O(Kn)$. □

The time T^* was discussed in Section 9.1. The most consuming part is the generation of P_k^k at lines 10-14 of COMPBS; this requires $O(n)$ time since it must copy $\underline{x}^{h^*}(k-1)$, $\bar{x}^{h^*}(k-1)$ and $D_{\pm}^{h^*}(k-1)$. The time requirement of this part will be further reduced in the next section by introducing a sophisticated data structure for P_k^h.

9.5 Time and Space Reduction

The time and space required for KBS will be reduced to $O(T^* + K \log K + \sqrt{n \log n})$ and $O(K\sqrt{n \log n})$ respectively in this section. To attain these bounds, it is essential to output x^k by $[i^*, j^*]$ and h^* (as indicated at line 4 of COMPBS). If n-dimensional vectors x^k's are to be directly output, $O(Kn)$ time is required only for this purpose.

9.5.1 Derivation tree T and some definitions

First we represent the derivation process of x^2, x^3, ... from x^1 by a rooted tree T defined as follows: x^{h^*} is the father of x^k (or x^k is a son of x^{h^*}) if x^k is obtained from x^{h^*} by an exchange. x^ℓ is an ancestor of x^h (or x^h is a descendant of x^ℓ) if there is a sequence $x^{\ell 1}$ ($= x^\ell$), $x^{\ell 2}$, ..., $x^{\ell p}$($=x^h$) such that $x^{\ell i}$ is the father of $x^{\ell i+1}$ (i=1, 2,..., p-1). x^h and $x^{h'}$ ($h \neq h'$) are brothers if their fathers coincide; x^h is placed to the left of $x^{h'}$ if $h < h'$. obviously, x^1 is the root of this tree. For an ancestor x^ℓ of x^h, $\pi(\ell, h)$ denotes the path from x^ℓ to x^h in T. For any x^p, let $[i^*(p), j^*(p)]$ denote the exchange with which x^p is obtained from its father.

Now consider the time instance when x^k is attached to x^{h^*} in T, corresponding to the generation of x^k from x^{h^*}. A special attention need be paid to the fact that $\bar{x}^{h^*}(k)$ of (9.5) is modified whenever its son is created. This implies that each $\bar{x}^h(k)$ is

generally dependent upon the entire history of how $x^1, x^2, \ldots, x^k$ have been generated (i.e., the structure of T). Let x^ℓ be an ancestor of x^h in T. Then we see from (9.5) that $\bar{x}^h(k)$ is computed from $\bar{x}^\ell(\ell)$ by taking into account the effects caused by those vertices in T which are either left brothers of some x^q on $\pi(\ell, h)$ or are sons of x^h. To carry out this computation, we define

$$f(h): x^{f(h)} \text{ is the father of } x^h, \tag{9.15}$$

$$s(h): x^{s(h)} \text{ is the rightmost son of } x^h \ (s(h) = \phi \text{ if } x^h \text{ has no son}), \tag{9.16}$$

$$b(h): x^{b(h)} \text{ is the brother immediately to the left of } x^h \ (b(h) = \phi \text{ if } x^h \text{ is the leftmost son}), \tag{9.17}$$

$$L_b(\ell, h) = \{i^*(p) \mid x^p \text{ is a left brother of some } x^q \text{ (note } p \neq q) \text{ on } \pi(\ell, h), \text{ and } q \neq \ell\}, \tag{9.18}$$

$$I_s(h) = \{i^*(p) \mid x^p \text{ is a son of } x^h\}, \tag{9.19}$$

$$J(\ell, h) = \{j^*(p) \mid x^p \text{ is on } \pi(\ell, h) \text{ and } p \neq \ell\}, \tag{9.20}$$

$$I(\ell, h) = \{i^*(p) \mid x^p \text{ is on } \pi(\ell, h) \text{ and } p \neq \ell\}. \tag{9.21}$$

Then we have, by the definition of T and (9.5),

$$x^h = x^\ell + \sum_{i \in I(\ell, h)} e_i - \sum_{j \in J(\ell, h)} e_j, \text{ where } e_q = (0, \ldots, 0, \overset{q}{1}, \ldots, 0), \tag{9.22}$$

$$\bar{x}_i^h(k) = \begin{cases} x_i^{p^*}, \text{ where } p^* = \max\{p \mid i^*(p) \in I(\ell, h) \text{ and } i^*(p) = i\} \\ \bar{x}_i^\ell(k), \text{ if } p^* \text{ is not defined,} \end{cases} \tag{9.23}$$

$$i = 1, 2, \ldots, n$$

$$\bar{x}_i^h(k) = \begin{cases} x_i^{f(p^*)}, \text{ where } p^* = \max\{p \mid i^*(p) \in I_b(\ell, h) \cup I_s(h) \text{ and } i^*(p)=i\} \\ \bar{x}_i^{\ell}(\ell), \text{ if } p^* \text{ is not defined,} \end{cases} \quad i = 1,2, \ldots, n. \tag{9.24}$$

9.5.2 Type 1 and type 2 data structures of P_k^h

Based on the above notations, we now introduce two types of data structures of P_k^h, with which the time and space reduction is attained. Call the following P_k^h type 1:

$$P_k^h = (c', [i', j'], [i^*(h), j^*(h)], x^h, \underline{x}^h(k), \bar{x}^h(k), D_-^h(k), D_+^h(k), \bar{x}^h(h), D_+^h(h), f(h), f'(h), b(h), s(h)), \tag{9.25}$$

where c' and [i', j'] are defined after (9.14) and f'(h) will be explained later in Subsection 9.5.3. Note that x^h is obtained from $x^{f(h)}$ by an exchange $[i^*(h), j^*(h)]$. A type 1 P_k^h requires O(n) space.

We also use a simplified data structure called type 2:

$$P_k^h = (c', [i', j'], [i^*(h), j^*(h)], f(h), f'(h), b(h), s(h)), \tag{9.26}$$

A type 2 P_k^h requires only constant space.

Consider now the instance immediately after x^k is obtained from x^{h^*}. It is shown in this and next subsections how a type 2 P_k^h (though only the cases of $h = h^*$, k are our concern because $P_k^h = P_{k-1}^h$ for $h \neq h^*$, k as discussed above) can be constructed from type 1 P_k^{ℓ}, where x^{ℓ} is an ancestor of x^h such that no vertex x^p on(ℓ, h) $(p \neq \ell)$ has type 1 P_k^p. The essential part is the computation of [i', j'] and c' of P_k^h, which is done after temporarily constructing $D_{\pm}^h(k)$.

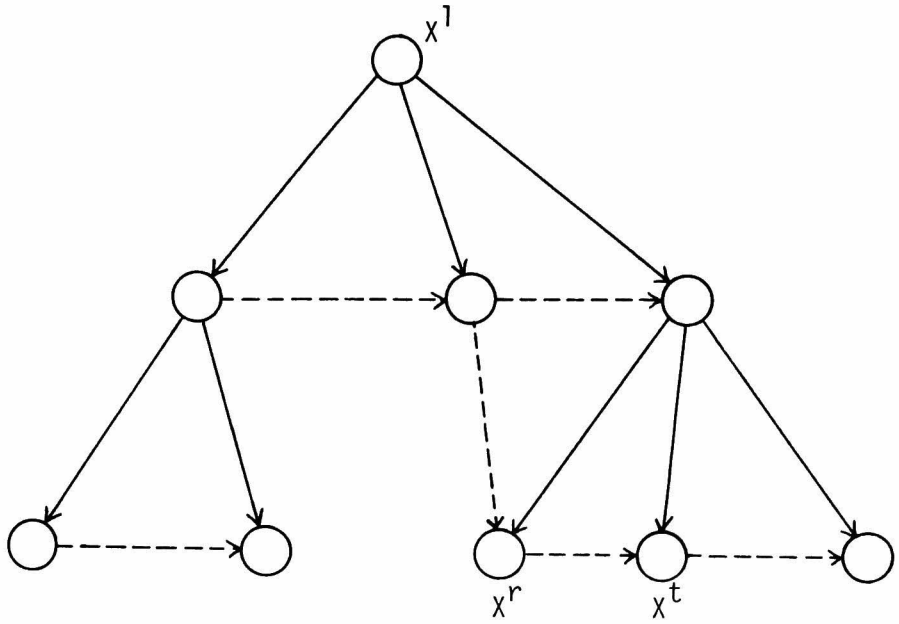

Fig. 9.1 The relation between a rooted tree T and the directed forest T' (solid lines denote T and broken lines denote T').

Based on (9.23) and (9.7), $D_{-}^{h}(k)$ is constructed in $O(|\pi(\ell, h)|$ log n) time as follows ($|\pi|$ denotes the length of path π). $\underline{x}^{h}(k)$ is obtained from $\underline{x}^{\ell}(k)$ by changing $x_{i*(p)}^{\ell}(k)$ to $x_{i*(p)}^{p}$ for each x^{p} on $\pi(\ell, h)$ by following $\pi(\ell, h)$ from x^{ℓ} to x^{h}. This requires $O(|\pi(h, \ell)|)$ time, since a change of an element is done in constant time. Corresponding to the changes in $\underline{x}^{h}(k)$, $D_{-}^{h}(k)$ is obtained by appropriately modifying $\underline{D}^{\ell}(k)$ in $O(|\pi(\ell, h)|\log n)$ time since a deletion or a change of an element in $D_{-}^{\ell}(k)$ is done in O(log n) time.

Note that, after type 2 P_{k}^{h} is computed (which will be explained in the next subsection), the modified $\underline{x}^{\ell}(k)$ and $\underline{D}^{\ell}(k)$ in P_{k}^{ℓ} (which now become $\underline{x}^{h}(k)$ and $D_{-}^{h}(k)$ respectively) must be set back to the original form. This is also done in $O(|\pi(\ell, h)|\log n)$ time just by following $\pi(\ell, h)$ reversely.

The construction of $D_{+}^{h}(k)$ from $D_{+}^{\ell}(k)$ is more involved and discussed in the next subsection.

9.5.3 Directed forest T' and construction of a type 2 P_{k}^{h}

To compute $D_{+}^{m}(k)$ efficiently, we introduce a directed forest T' defined as follows (see Fig. 9.1): T' has vertices x^{1}, x^{2}, ..., x^{k-1}, and x^{r} is the father of x^{t} in T' (denoted by $r = f'(t)$) if $r = b(t)$, or if $b(t) = \phi$ and $r = b(q)$ where x^{q} is the closest ancestor of x^{t} in T with $b(q) \neq \phi$. Note that an x^{p} is a root in T' if and only if x^{p} is on the leftmost path in T. The path from x^{u} to x^{v} in T'

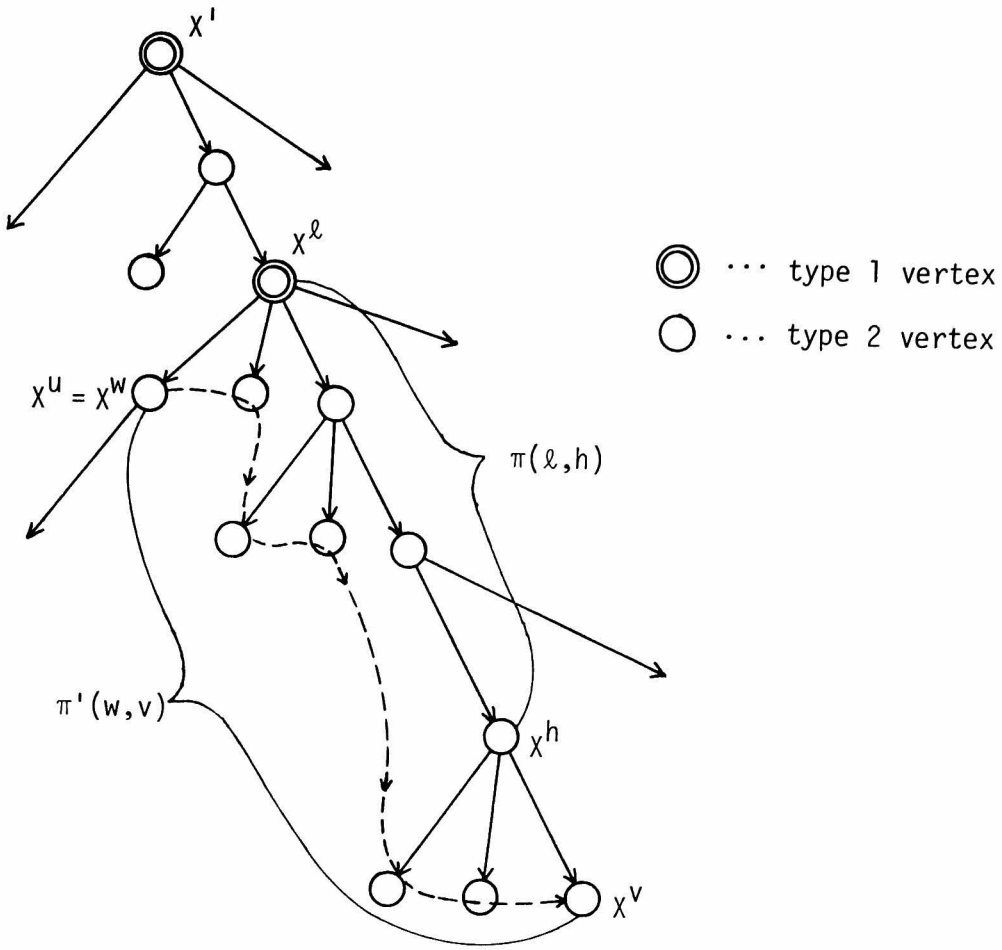

Fig. 9.2 Relative positions of the vertices in T and T' used for the computation of type 2 P_k^h.

(i.e., x^u is an ancestor of x^v in T') is denoted by $\pi'(u, v)$. Then we have by (9.18) and (9.19) that

$$I_b(\ell, h) \cup I_s(h) = \{i^*(p) \mid x^p \text{ is on } \pi'(u, v)\}, \tag{9.27}$$

where

$$v = \begin{cases} s(h), \text{ if } s(h) \neq \phi \text{ (see (9.16))} \\ b(q), \text{ otherwise} \end{cases}$$

$$q = \max\{p \mid x^p \text{ is on } \pi(\ell, h),\ p \neq \ell \text{ and } b(q) \neq \phi\}.$$

$$u = \min\{r \mid x^r \text{ is an ancestor of } x^v \text{ in T' and is a descendant of } x^\ell \text{ in T}\}.$$

The situation is illustrated in Fig. 9.2. If x^u and x^v are not defined by this (i.e., the set on the right hand side of q is empty), $I_b(\ell, h) \cup I_s(h) = \phi$ is concluded. x^v can be computed in $O(|\pi(\ell, h)|)$ time. Let

$$w = \begin{cases} \max\{q \mid x^q \text{ is on } \pi'(u, v) \text{ and is of type 1}\} \\ u \text{ if type 1 vertex is not on } \pi'(u, v). \end{cases} \tag{9.28}$$

x^w is obtained in $O(|\pi'(w, v)|)$ time by following T' upward from x^v, checking at each vertex x^r whether $f(r) \geq \ell$ (i.e., x^r is a descendant of x^ℓ in T) or not.

Case A: If x^w is of type 2 (i.e., $w = u$), $\bar{x}^h(k)$ and $D_+^h(k)$ are obtained as follows. Starting with x^ℓ, $\bar{x}^\ell(\ell)$ and $D_+^\ell(\ell)$ in $P^\ell(k-1)$ are modified according to (9.5) and (9.9) so as to take into account the effects of the sons of x^ℓ which is in $\pi'(w, v)$. Then move to

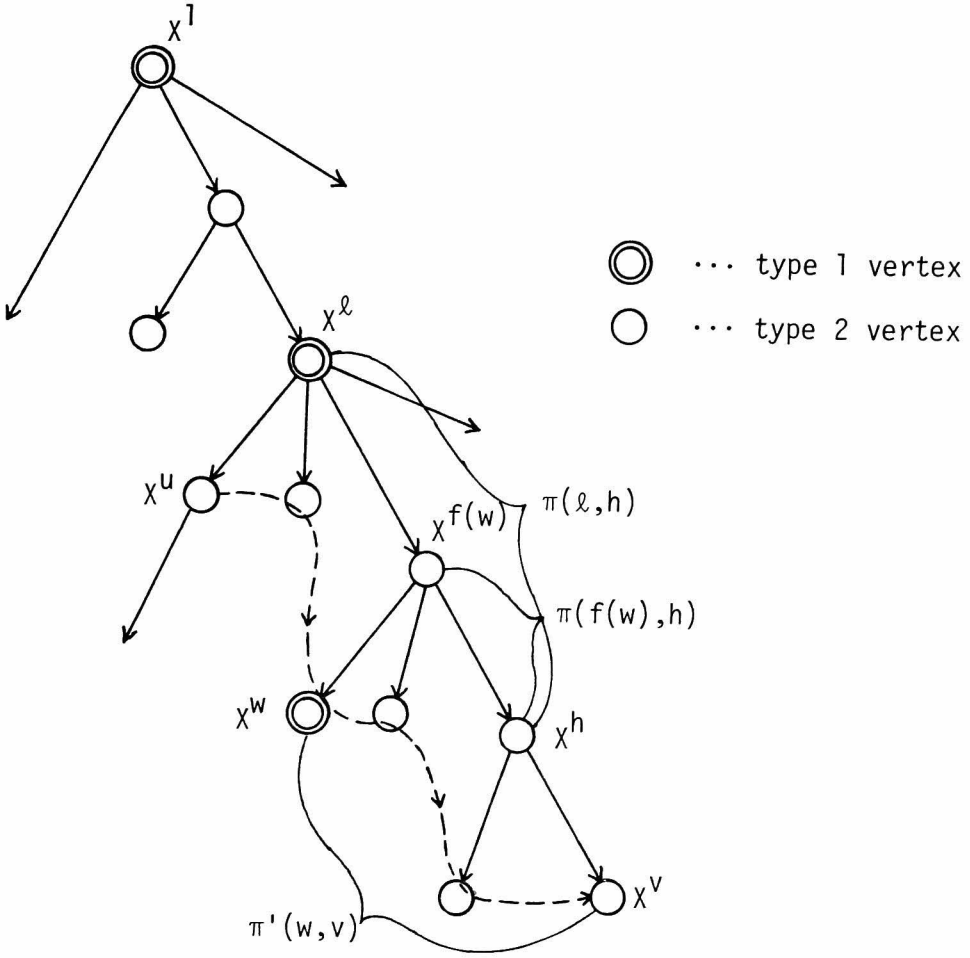

Fig. 9.3 The relative position of the vertices in T and T' when $\pi'(u,v)$ has a type 1 vertex.

the son of x which is in $\pi(\ell, h)$, and modify the above uptated $\bar{x}^{\ell}(\ell)$ and $D_{+}^{\ell}(\ell)$ by (9.5) and (9.11). The abve computation is repeated until x^{h} is reached. Since each above operation requires $O(\log n)$ time, the total required time is $O(|\pi(f(w), h)|\log n + |\pi'(w, v)|\log n)$.

Case B: If x^{w} is of type 1 (see Fig. 9.3), we have

$$\bar{x}_i^{h}(k) = \begin{cases} x_i^{f(p^*)}, \text{ where } p^* = \max\{p \mid p(\neq w) \text{ is on } \pi'(w,v) \\ \qquad\qquad\qquad\qquad \text{and } i^*(p) = i\}. \\ \bar{x}_i^{f(w)}(w), \text{ if } p^* \text{ is not defined,} \end{cases} \tag{9.29}$$

by (9.24) and (9.27). First $\bar{x}^{f(w)}(w)$ and $D_{+}^{f(w)}(w)$ are obtained in $O(\log n)$ time from $\bar{x}^{w}(w)$ and $D_{+}^{w}(w)$ by using (9.5), (9.9) and (9.11). Then $\bar{x}^{h}(k)$ and $D_{+}^{h}(k)$ are obtained in $O(|\pi(f(w), h)|\log n + |\pi'(w, v)|\log n)$ time from $\bar{x}^{f(w)}(w)$ and $D_{+}^{f(w)}(w)$ in a manner similar to Case A.

Upon constructing $D_{+}^{h}(k)$ and $D_{-}^{h}(k)$, it is easily seen that c' and [i', j'] can be computed in $O(\log n)$ time as noted in Section 9.3 after (9.8). Other data [i*(h), j*(h)], f(h), b(h) and s(h) in Type 2 $P^{h}(k)$ can be obtained in constant time as follcws: For $h = k$,

$$\begin{aligned} &[i^*(k), j^*(k)] \leftarrow [i', j'] \text{ (in } P_{k-1}^{h^*}), \\ &f(k) \leftarrow h^*, \; f'(k) \leftarrow \begin{cases} s(h^*), \text{ if } s(h^*) \neq \phi \\ f'(h^*), \text{ otherwise} \end{cases} \\ &b(k) \leftarrow s(h^*), \; s(k) \leftarrow \phi. \end{aligned} \tag{9.30}$$

For $h = h^*$,

$$s(h^*) \leftarrow k \tag{9.31}$$

and other data in type 2 $P_{k-1}^{h^*}$ do not change. From the above discussion the next lemma follows.

Lemma 9.6 A type 2 P_k^h is constructed in $O(|\pi(f(w), h)|\log n + |\pi'(w, v)|)$ time, where x^v, x^w and x^h are defined as above. A type 2 P_k^h requires constant space. □

9.5.4 Scheme of creating type 1 data structure

Assume that P_1^1 is of type 1 and all the other P_k^h's are initially type 2. A P_k^h $(h > 1)$ is then altered from type 2 to type 1 if one of the following three conditions is satisfied.

(a) $|\pi(\ell, h)| = y$ and

$$\text{(the number of descendants of } x^h \text{ in T)} \geq y \tag{9.32}$$

hold where y is a prespecified positive integer not larger than K and x^ℓ is the closest ancestor of x^h in T with type 1 data structure.

(b) x^h is on the leftmost path of T (i.e., a root in T'), and

$$\text{(the number of descendants of } x^h \text{ in T')} \geq y. \tag{9.33}$$

(c) x^h is not on the leftmost path of T, and satisfies

$$|\pi'(w, h)| = y \text{ and (9.33)}, \tag{9.34}$$

where x^w is the closest ancestor of x^h in T' with type 1 data structure. (Note that if an x^h of (c) satisfies (9.33), there is an ancestor in T' satisfying the condition of (b), and therefore the above x^w always exists.)

Consequently it always holds for any x^h that

$$|\pi(\ell, h)| \leq 2y \tag{9.35}$$

$$|\pi'(w, h)| \leq 2y. \tag{9.36}$$

Lemma 9.7 The numbers of P_k^h of type 1 and type 2 are $O(K/y)$ and $O(K)$ respectively.

Proof. The result for type 2 is obvious. We show that the number of type 1 P_k^h is $O(K/y)$. Let type 1a, 1b and 1c denote the type 1 data structure generated by the above rules (a), (b) and (c) respectively. Consider P_1^1 be of type 1a for convenience. Define a set

$$D(h) = \{x^p \mid x^p \text{ is a vertex not of type 1a, whose closest ancestor in T is } x^h\},$$

for a type 1a P_k^h. By the generation rule (9.32), D(h) contains at least y-1 vertices. Since D(h)'s for type 1a vertices x^h are mutually disjoint and their union is the set of all vertices not of type 1a, the number of type 1a vertices is $O(K/y)$. It is similarly shown that the number of type 1b or 1c vertices is $O(K/y)$. Thus the total number of type 1 vertices is $O(K/y)$. □

9.5.5 Time to create type 1 data structure

We analyze here the time to create type 1 data. It is not difficult to see that the conditions (9.32)-(9.34) to alter type 2 to type 1 can be detected in constant time, by introducing some additional data such as the numbers of descendants of x^h in T and T' respectively. When it is decided to change type 2 P_k^h to type 1, let x^ℓ be the closest type 1 ancestor of x^h in T, and let x^v and x^w be defined by (9.27) and (9.28) (see Fig. 9.2). As discussed in Subsections 9.5.2 and 9.5.3, x^h, $\underline{x}^h(k)$, $D_-^h(k)$, $\bar{x}^h(k)$ and $D_+^h(k)$ are obtained in $O(y \log n + n)$ time (including the time to copy these data) since $|\pi(f(w), h)| \leq |\pi(\ell, h)| \leq 2y$ and $|\pi'(w, v)| \leq 2y$ hold by (9.35) and (9.36). $\bar{x}^h(h)$ and $D_+^h(h)$ are also obtained in $O(y \log n + n)$ time from $\bar{x}^h(k)$ and $D_+^h(k)$ by a similar procedure. The other data can be obtained in constant time by (9.30). Consequently the following lemma is proved.

Lemma 9.8 The time required to alter type 2 P_k^h to type 1 is $O(y \log n + n)$. □

The results of Lemmas 9.6-9.8 are summarized in Table 9.1.

type	time for a P_k^h	space for a P_k^h	total number
1	$O(n+y \log n)$	$O(n)$	$O(K/y)$
2	$O(y \log n)$	constant	$O(K)$

Table 9.1. The time and space requirement by the modified data structure

9.5.6 Time and space complexity of the entire algorithm

This section analyzes the time and space requirement of Algorithm KBS and Subroutine COMPBS implemented with the above data structure. First consider the time requirement of COMPBS for each iteration. The time required for lines 5-14 is reduced to $O(y \log n)$ by Lemma 9.6 and (9.35), (9.36). The other lines are not changed. Thus COMPBS requires $O(y \log n + \log K)$ time for every iteration. Since COMPBS is called K-1 times, the total time is

$$O(K) \cdot O(y \log n + \log K) = O(Ky \log n + K \log K). \qquad (9.37)$$

In addition to the time consumed by COMPBS, the modified KBS with the new data structure must take care of the alterations of some P_k^h from type 2 to type 1. The total time required for this process is

$$O(K/y)\cdot O(y \log n + n) = O(K \log n + Kn/y) \qquad (9.38)$$

by Lemmas 9.7 and 9.8. Thus the modified KBS requires

$$O(Ky \log n + K \log K + Kn/y)$$

in total.

The space requirement for all P_k^h is also easily obtained from Table 9.1;

$$O(K) + O(K/y)\cdot O(n) = O(K + Kn/y).$$

Other space is obviously dominated by this.

Letting $y = \min(K, \sqrt{n/\log n})$ in the above discussion results in the next theorem.

Theorem 9.9 KBS (with subroutine COMPBS) can be implemented with $O(T^* + K \log K + K\sqrt{n \log n})$ time and $O(K\sqrt{n \log n} + n)$ space, where T^* is the time to obtain x^1. □

The $+n$ in the space complexity is added since at least $O(n)$ space in necessary even if $K < \sqrt{n/\log n}$. It is not added to the time complexity because T^* is at least $O(n)$.

9.6 Conclusion

This chapter has first proposed an efficient algorithm for obtaining K best solutions of the simple resource allocation problem in Section 9.4. This requires $O(T^* + K\log K + Kn)$ time and $O(Kn)$ space. Then more efficient algorithm has been proposed in Section 9.5, which requires $O(T^* + K\log K + K\sqrt{n\log n})$ time and $O(K\sqrt{n\log n} + n)$ space. The latter algorithm is quite efficient, but it is difficult to implement, due to the rather complicated data structures. Therefore, if K or n is comparatively small, it might be better to use the former algorithm. Finally, it should be noted that, by applying almost the same idea as developed in this chapter, it is also possible to develop an efficient algorithm for obtaining K best solutions of the problem discussed in Chapter 6.

CHAPTER 10

CONCLUSION

Throughout the thesis, we have developed efficient algorithms for several types of combinatorial optimization problems. The thesis emphasizes the following two points:

(i) The efficient algorithms have been developed to obtain K best solutions for the famous two graph optimization problems, i.e., the shortest path problem and the minimum spanning tree problem. Both algorithms have been shown to be superior to any of the existing algorithms.

(ii) Several types of resource allocation problems have been considered. The first problem is to minimize the sum of single-variable separable convex functions under the only one resource constraint saying that the sum of integer variables is equal to a given integer. All of the other resource allocation problems discussed in this thesis are generalizations or variants of the first problem. The second problem is obtained by allowing more than one resource constraint. The third one is obtained by interchanging the objective function with the constraint function. The fourth one has the objective function which is to minimize the maximum of the profit differences between each pair of activities. The final problem is to obtain K best solutions of the first problem. We have proposed efficient algorithms for

all these problems except for the second one, which we have shown to be intrinsically difficult.

The proposed efficient algorithms will be useful and important from both theoretical and practical points of view. In addition, they may clarify the problem structure.

Many of the real life problems can be formulated as combinatorial optimization problems. Importance of the development of efficient algorithms for the combinatorial optimization problems has been and will be continuously increasing. The author hopes that the work contained in this thesis will move the status of combinatorial optimization techniques one step forward.

APPENDIX

In this appendix we summarize some graph-theoretical definitions and notations necessary to read the thesis. The reader should refer to Harary [H1] for the details.

A graph G = (V, E) is an ordered pair consisting of a set V of vertices and a set E of edges. Either the edges are ordered pairs (v, w) of distinct vertices (the graph is directed), or the edges are unordered pairs of distinct vertices, also represented as (v, w) (the graph is undirected). If (v, w) is an edge, v and w are its endpoints and are adjacent. A graph $G' = (V', E')$ is a subgraph of G if $V' \subseteq V$ and $E' \subseteq E$. G' is spanning if $V' = V$.

A path from v_1 to v_n in G is a requence of edges (v_1, v_2), $\cdots$, (v_{n-1}, v_n). This path is said to contain edges (v_1, v_2), $\cdots$, (v_{n-1}, v_n) and vertices $v_1, \cdots, v_n$, and to avoid all other edges and vertices. The path is simple if $v_1, \cdots, v_n$ are distinct except possibly v_1 and v_n; the path is a cycle if $v_1 = v_n$.

An undirected graph is connected if there is a path from any vertex to any other vertex. A directed graph is strongly connected if there is a path from any vertex to any other vertex. The maximum connected (strongly connected) subgraphs of a graph are called its connected components (strongly connected components).

A tree T is a connected undirected graph which contains no cycles. A forest is a set of trees. In a tree there is a unique

simple path between any pair of distinct vertices. A rooted, undirected (T, r) is a tree with a distinguished vertex r, called the root. A rooted, directed tree (an arborescence) is a directed graph T with a unique vertex r such that

(i) there is a path from r to any other vertex;

(ii) each vertex except r has exactly one edge leading to it;

(iii) r has no edges leading to it.

Any rooted, undirected tree (T, r) can be converted into a rooted, directed tree by directing each edge (v, w) so that v is contained in the path from r to w.

In a rooted, directed tree, a vertex w is a descendant of a vertex v (v is an ancestor of w) if there is a path from v to w. A vertex v is a son of v (v is the father of w) if (v, w) is an edge in the tree. These definitions extend to rooted, undirected trees by directing the edges of the tree as above. If G is a graph, a spanning tree of G is a rooted tree which is a spanning subgraph of G.

A partition $\mathcal{P}$ of a set S is a collection of subsets $S_1, S_2, \cdots, S_k$ of S such that $\bigcup_{i=1}^{k} S_i = S$ and $S_i \cap S_j = \phi$ for $i \neq j$. If $\mathcal{P}$ and $\mathcal{P}'$ are partitions of S, $\mathcal{P}$ is a refinement of $\mathcal{P}'$ (and is a coarsening of $\mathcal{P}$) if, for all $S_i \in \mathcal{P}$, there is some $S_j' \in \mathcal{P}'$ such that $S_i \subseteq S_j'$.

REFERENCES

[A1] A. V. Aho, J. E. Hopcroft and J. D. Ullman, The Design and Analysis of Computer Algorithms, Addison-Wesley, Reading, MA., 1974.

[B1] M. L. Balinski and H. P. Young, The quota method of apportionment, Amer. Math. Monthly, 82 (1975), pp. 701-730.

[B2] _____, On Huntington method of apportionment, SIAM. J. Appl. Math., 33 (1977), pp. 607-618.

[B3] _____, Criteria for proportional representation, Operations Research, 27 (1979), pp. 80-95.

[B4] R. Bellman and R. Kalaba, On k-th best policies, J. Soc. Indust. Appl. Math., 8 (1960), pp. 582-588.

[B5] _____, Dynamic Programming, Princeton University Press, Princeton N.J., 1957.

[B6] M. Blum et al., Time bounds for selection, J. Computer and System Sciences, 7 (1973), pp. 448-461.

[B7] J. R. Brown, The knapsack sharing problem, Operations Research, 27 (1979), pp. 341-355.

[B8] R. N. Burns and C. E. Haff, A ranking problem in graphs, Proceedings of 5th South-East Conf. Combinatorics, Graph Theory and Computing, 19 (1974), pp. 461-470.

[C1] P. M. Camerini, L. Fratta and F. Maffioli, The K shortest spanning trees of a graph, Int. Rep. 73-10, IEE-LCE, Politechnico di Milano, Italy, 1974.

[C2] P. M. Camerini, L. Fratta and F. Maffioli, The K best spanning arborescences of a network, to appear in Networks.

[C3] D. Cheriton and R. E. Tarjan, Finding minimum spanning trees, SIAM J. Comput., 5 (1976), pp. 724-742.

[C4] S. Clarke, A. Krikorian and J. Rausen, Computing the N best loopless paths in a network, J. SIAM, 11 (1973), pp. 269-271.

[C5] S. A. Cook, The complexity of theorem proving procedures, Third ACM Symp. on Theory of Computing, (1971), pp. 151-158.

[D1] E. W. Dijkstra, A note on two problems in connexion with graphs, Numerische Mathematik, 1 (1959), pp. 269-271.

[D2] S. E. Dreyfus, An appraisal of some shortest path algorithms, Operations Research, 17 (1965), PP. 395-412.

[D3] _____ and A. M. Law, The Art and Theory of Dynamic Programming, Academic Press, New York, 1977.

[D4] F. D. J. Dunstan, An algorithm for solving a resource allocation problem, Operational Research Quarterly, 28 (1977), pp. 839-851.

[E1] J. M. Einbu, On Shih's incremental method in resource allocations, Operational Research Quarterly, 28 (1977), pp. 459-462.

[E2] _____, Optimal allocations of continuous resources to several activities with a concave return function - some theoretical results, Mathematics of Operations Res., 3 (1978), pp. 82-88.

[F1] B. Fox, Discrete optimization via marginal analysis,

Management Science, 13 (1966), pp. 210-216.

[F2] B. Fox, Calculating Kth shortest paths, INFOR., 11 (1973), pp. 66-70.

[F3] L. R. Ford and D. R. Fulkerson, Flows in Networks, Princeton University Press, Princeton, N.J., 1962.

[F4] G. N. Frederickson and D. B. Johnson, Optimal algorithms for generating quantile information in X + Y and matrices with sorted columns, Technical Report CS-79-45, Computer Science Dept., The Pennsylvania State University 1979.

[G1] H. N. Gabow, Two algorithms for generating weighted spanning trees in order, SIAM J. Comput., 6 (1977), pp. 139-150.

[G2] Z. Galil and N. Meggido, A fast selection algorithm and the problem of optimum distribution of effort, J. ACM, 26 (1979), pp. 58-64.

[G3] M. R. Garey and D. S. Johnson, Strong NP-completeness results: motivation, examples and implications, J. ACM, 25 (1978), pp. 499-508.

[G4] _____, Computers and Intractability: A Guide to the Theory of NP-Completeness, W. H. Freeman and Company, San Franscisco 1979.

[G5] O. Gross, A class of discrete-type minimization problems, RM-1644-PR, The RAND Corporation, Santa Monica, California, 1956.

[H1] F. Harary, Graph Theory, Addison-Wesley, Reading, MA., 1969.

[H2] R. Hartley, On an algorithm proposed by Shih, Operational Research Quarterly, 27 (1976), pp. 389-390.

[H3] M. Held and R. M. Karp, The travelling-salesman problem and minimum spanning trees, Operations Research, 18 (1970), pp. 1138-1162.

[H4] _____, The travelling-salesman problem and minimum spanning trees: Part II, Mathematical Programming, 1 (1971), pp. 6-25.

[H5] W. Hoffman and R. Paveley, A method for the solution of the Nth best path problem, J. ACM, 6 (1959), pp. 506-514.

[I1] T. Ibaraki, Finite state representations of discrete optimization problems, SIAM J. Comput., 2 (1973), pp. 193-210.

[I2] _____, Classes of discrete optimization problems and their decision problems, J. Computer and System Sciences, 8 (1974), pp. 84-116.

[I3] _____, N. Katoh and H. Mine, Obtaining K best solutions in combinatorial optimization problems, 10th International Symp. on Mathematical Programming, Montreal, 1979.

[I4] _____, Resource allocation with a convex objective function and its generalizations, to be presented at Summer School on Generalized Concavity in Optimization and Economics, NATO Advanced Study Institute, August 1980.

[I5] _____, Theoretical comparisons of search strategies in branch-and-bound algorithms, International J. Computer

and Information Sciences, 5 (1976), pp. 315-344.

[I6] T. Ibaraki, Computational efficiency of approximate branch-and-bound algorithms, Mathematics of Operations Res., 1 (1976), pp. 287-298.

[I7] _____, On the computational efficiency of branch-and-bound algorithms, J. Operations Research Society of Japan, 20 (1977), pp. 16-35.

[I8] _____, Branch-and-bound procedure and state-space representation of combinatorial optimization problems, Information and Control, 36 (1978), pp. 1-27.

[I9] _____, Theory of Combinatorial Optimization, IECEJ and Corona-sha, Tokyo, 1979.

[I10] S. Ikeda, Y. Nakamori and Y. Sawaragi, A measure of obtaining representative pointwise source and its availability for air pollution control, International J. System Sciences, 8 (1977), pp. 1429-1440.

[J1] S. Jacobsen, On marginal allocation in single constraint min-max problems, Management Science, 17 (1971), pp. 780-783.

[J2] S. M. Johnson, Sequential product planning over time at minimum cost, Management Scinece, 3 (1957), pp. 435-437.

[J3] D. B. Johnson, Efficient algorithms for shortest paths in sparse networks, J. ACM, 24 (1977), pp. 1-13.

[K1] E. P. Kao, On incremental analysis in resource allocations, Operational Research Quarterly, 27 (1976), pp. 759-763.

[K2] R. Karp and M. Held, Finite-state processes and dynamic programming, SIAM J. Applied Mathematics, 15 (1967), pp. 693-718.

[K3] _____, Reducibility among Combinatorial Problems, Complexity of Computer Computations, R. E. Miller and J. W. Thatcher, Eds., Plenum Press, New York, 1972, pp. 85-104.

[K4] W. Karush, On a class of minimum-cost problems, Management Science, 4 (1958), pp. 136-153.

[K5] N. Katoh, T. Ibaraki and H. Mine, An $O(Kn^2)$ algorithm for K shortest simple paths in an undirected graph with non-negative arc lenght, (in Japanese) The Transactions of the IECE of Japan, D 61 (1978), pp. 1199-1206.

[K6] _____, A polynomial time algorithm for the resource allocation problem with a convex objective function, J. Operational Research Society, 30 (1979), pp. 449-455.

[K7] _____, Algorithms for a variant of the resource allocation problem, J. Operations Research Society of Japan, 22 (1979), pp. 287-300.

[K8] _____, An algorithm for finding K minimum spanning trees, to appear in SIAM J. on Computing.

[K9] _____, Notes on the problem of the allocation of resources to activities in discrete quantities, Journal of Operational Research Society, 31 (1980), pp. 595-598.

[K10] N. Katoh, T. Ibaraki and H. Mine, Algorithms for a variant of the resource allcation problem, 10th International Symp. on Mathematical Programming, Montreal, 1979.

[K11] _____, An $O(Kn^2)$ algorithm for K shortest simple paths with nonnegative edge length, submitted to SIAM J. Applied Mathematics.

[K12] _____, An algorithm for K best solutions of the resource allocation problem, to appear in J. ACM.

[K13] _____, Equipollent resource allocation problem with application to optimal apportionment, submitted to Operations Research.

[K14] _____, An algorithm for an equipollent resource allocation problem, submitted to Mathematics of Operations Research.

[K15] V. Klee and G. J. Minty, How good is the simplex method?, Inequalities-III, O. Shisha, ed., Academic Press, New York, (1972), pp. 159-175.

[K16] D. E. Knuth, The Art of Computer Programming Vol. 3: Sorting and Searching, Addison-Wesley, Reading, MA., 1973.

[K17] B. O. Koopman, The optimum distribution of effort, Operations Research, 1 (1952), pp. 52-63.

[K18] J. Kruskal, On the shortest spanning subtree of a graph and the travelling salesman problem, Proc. Amer. Math. Soc.,

2 (1956), pp. 48-50.

[L1] E. L. Lawler, A rpocedure for computing the K best solutions to discrete optimization problems and its application to the shortest path problems, Management Science, 18 (1972), pp. 401-405.

[L2] _____ and D. E. Wood, Branch-and-bound methods: A survey, Operations Research, 14 (1966), pp. 699-719.

[L3] _____, Comment on computing the K shortest paths in a graph, C. ACM, 20 (1977), pp. 603-604.

[L4] _____, Combinatorial Optimization: Networks and Matroids, Holt, Rinehart and Winston, New York, 1976.

[L5] G. S. Lueker, Two polynomial complete problems in nonnegative integer programming, Computer Science TR-178, Princeton Univ., Princeton, N. J., 1975.

[L6] H. Luss and S. K. Gupta, Allocation of effort resources among competing activities, Operations Research, 23 (1975), pp. 360-366.

[M1] I. Michaeli and M. A. Pollatscheck, On some nonlinear knap- problems, Ann. Disc. Math., 1 (1977), pp. 403-414.

[M2] E. Minieka, On computing sets of shortest paths in a graph, C. ACM, 17 (1974), pp. 351-353.

[M3] _____ and D. Shier, A note on an algebra for the K best routes in a network, J. Inst. Math. Appl., 11 (1973), pp. 145-149.

[M4] K. M. Mjelde, The optimality of an incremental solution of a problem related to distribution of effort, Operational Research Quarterly, 26 (1975), pp. 867-870.

[M5] _____, Evaluation and incremental determination of almost optimal allocations of resources, Operational Research Quarterly, 27 (1976), pp. 581-588.

[M6] K. G. Murty, An algorithm for ranking all the assignments in increasing order of cost, Operations Research, 16 (1968), pp. 682-687.

[N1] G. Nemhauser and R. Garfinkel, Integer Programming, John Wiley, New York, 1972.

[P1] M. Pollack, The Kth best route through a network, Operations Research, 9 (1961), pp. 578-579.

[P2] _____, Solutions of the kth best route through a network - a review, J. Math. Anal. and Appl., 3 (1961), pp. 547-559.

[P3] _____, Shortest path solutions of the kth best route problems, the 36-th National Meeting of the ORSA, Miami Florida, 1969.

[P4] E. L. Porteus and J S. Yormark, More on min-max allocation, Management Science, 18 (1972), pp. 502-507.

[P5] R. C. Prim, Shortest connection networks and some generalizations, Bell System Tech. J., 36 (1957), pp. 1389-1401.

[P6] L. G. Proll, Marginal analysis in resource allocations, Operational Research Quarterly, 27 (1976), pp. 765-767.

[R1] T. Rockafellar, Convex Analysis, Princeton Univ. Press, N. J., 1970.

[S1] T. L. Saaty, Mathematical Methods of Operations Research, McGraw-Hill, New York, 1959.

[S2] M. Sakarovitch, The K shortest routes and the K shortest chains in a graph, ORC 66-32, Operations Research Center, University of California, Berkeley, October, 1966.

[S3] D. R. Shier, Computational experience with an algorithm for finding the K shortest paths in a network, to appear in J. Res. Nat. Bur. Stand.

[S4] _____, Iterative methods for determining the K shortest paths in a network, Networks, 6 (1976), pp. 205-230.

[S5] W. Shih, A new application of incremental analysis in resource allocations, Operational Research Quarterly, 25 (1974). pp. 587-597.

[S6] _____, A branch and bound procedure for a class of discrete resource allocation problems with several constraints, Operational Research Quarterly, 28 (1977), pp. 439-451.

[S7] B. D. Sivazlian and L. E. Stanfel, Optimization Techniques in Operations Research, Prentice-Hall, Englewood Cliffs, 1975, pp. 432-434.

[T1] R. E. Tarjan, Depth-first search and linear graph algorithms, SIAM J. Computing, 1 (1972), pp. 146-160.

[T2] R. E. Tarjan, Efficiency of a good but not linear disjoint set union algorithm, J. ACM, 22 (1975), pp. 215-225.

[T3] _____, Applications of path compression on balanced trees, Tech. Rep. STAN-CS-75-512, Computer Science Dept., Stanford Univ., Stanford, CA., 1975.

[T4] _____, Complexity of combinatorial algorithms, SIAM Review, 20 (1978), pp. 457-491.

[T5] S. Tomizawa, An efficient algorithm for solving the shortest route problems, (in Japanese) The Transactions of IECE of Japan, J59-A (1976), pp. 523-530.

[V1] A. F. Veinott, Production planning with convex costs: A parametric study, Management Science, 10 (1964), pp. 441-460.

[V2] _____, The status of mathematical inventory theory, Management Science, 12 (1966), pp. 745-777.

[W1] H. M. Wagner, Principles of Operations Research, Prentice-Hall, Englewood Cliffs, 1969.

[W2] M. M. Weigand, Ein neuer algorithmus zur bestimung von k-kürzesten wegen in einem graphen, Computing, 16 (1976), pp. 139-151.

[W3] I. J. Weinstein and O. S. Yu, Comment on an integer maximization problem, Operations Research, 21 (1973), pp. 648-649.

[W4] A. Weintraub, The shortest and the K-shortest routes as assignment problems, Networks, 3 (1973), pp. 61-73.

[W5] A. Wongseelashote, An algebra for determining all path-values in a network with application to K-shortest path problems, Networks, 6 (1976), pp. 307-334.

[Y1] A. C. Yao, An O(|E| log log|V|) algorithm for finding minimum spanning trees, Information Processing Letters, 4 (1975), pp. 21-23.

[Y2] J. Y. Yen, Finding the K shortest loopless paths in a network, Management Science, 17 (1971), pp. 712-716.

[Y3] _____, Another algorithm for finding the K shortest loopless network paths, 41st National ORSA Meeting in New Orleans, April, 1972.